N° d'enregistrement C.N.R.S.
A.O 8065

THESE

DE

DOCTORAT D'ETAT ES-SCIENCES MATHEMATIQUES

présentée

A L'UNIVERSITÉ PARIS VI

par

Claude KAISER

pour obtenir

le grade de Docteur Es-Sciences

CONCEPTION ET RÉALISATION DE SYSTEMES A ACCES MULTIPLE : GESTION DU PARALLÉLISME

Soutenue le 13 février 1973 devant la Commission d'examen

MM. J. ARSAC *Président*
C. PAIR
P. FEAUTRIER } *Examinateurs*
H. BOUCHER

N° d'enregistrement C.N.R.S.
A.O 8065

THESE

DE

DOCTORAT D'ETAT ES-SCIENCES MATHEMATIQUES

présentée

A L'UNIVERSITÉ PARIS VI

par

Claude KAISER

pour obtenir

le grade de Docteur Es-Sciences

CONCEPTION ET RÉALISATION DE SYSTEMES A ACCES MULTIPLE : GESTION DU PARALLÉLISME

Soutenue le 13 février 1973 devant la Commission d'examen

MM. J. ARSAC *Président*
C. PAIR
P. FEAUTRIER } *Examinateurs*
H. BOUCHER

UNIV.
LEIDEN
BIBL.

Je tiens à remercier :

Monsieur ARSAC, Directeur de l'Institut de programmation, qui m'a fait l'honneur d'accepter de présider le jury de thèse,

Monsieur PAIR qui a bien voulu examiner ce travail,

Monsieur FEAUTRIER qui a accepté de participer au jury,

Monsieur BOUCHER qui a défini les projets Haliotis et Esope, dans le cadre desquels ce travail a été réalisé,

Tous mes collègues du centre de calcul Coelacanthe, du Laboria et du centre de calcul de l'Iria, qui m'ont aidé dans la réalisation de ma tâche, grâce à leurs remarques, leurs conseils ou leur contribution active,

J'exprime une reconnaissance particulière à Madame LETAILLIEUR et à Madame PEREZ, qui ont assuré la réalisation matérielle de cet ouvrage.

CONCEPTION ET REALISATION DE SYSTEME A ACCES MULTIPLE :

GESTION DU PARALLELISME

C. KAISER

TABLE DES MATIERES

Introduction

Première partie. Le système Haliotis

chapitre 1 - Présentation du système page : 9

1.1 le matériel

1.2 les fonctions

1.3 organisation du système

1.4 les travaux de la situation tactique

chapitre 2 - Les problèmes fondamentaux 19

2.1 gestion du parallélisme des travaux

2.2 répartition des ressources

Deuxième partie. Mécanismes de base du système ESOPE

chapitre 1 - Présentation du système 30

1.1 le matériel

1.2 les fonctions

1.3 les principes

chapitre 2 - Coopération des processus 40

2.1 définition des sémaphores

2.2 exemples d'emploi des sémaphores

2.3 réalisation des mécanismes

2.4 liste des principales primitives

chapitre 3 - Accès à l'information, mémoire virtuelle 65

3.1 introduction

3.2 mémoire virtuelle et pagination sur le 10070

3.3 mémoire virtuelle et usager

3.4 le couplage

3.5 fichiers

chapitre 4 - L'attribution des ressources 85

4.1 position du problème

4.2 présention du blocage

4.3 utilisation de la mémoire par les programmes

4.4 comportement des usagers conversationnels

Troisième partie. Utilisation des mécanismes de gestion du parallélisme

chapitre 1 - Conception et réalisation de l'allocateur 118

1.1 principes de l'allocation de ressource

1.2 circulation des usagers

1.3 les processus de l'allocateur

1.4 mécanismes et tables des données

chapitre 2 - Communication entre l'usager conversationnel et le système 170

2.1 généralités

2.2 admission et départ d'un utilisateur conversationnel

2.3 langage de commande

2.4 signalement des erreurs

2.5 le client ESOPE

2.6 le processus premier de l'usager temps réel

2.7 les processus de la communication usager-système

chapitre 3 - La gestion des périphériques 189

3.1 introduction

3.2 les mécanismes utilisés

3.3 schéma de réalisation du SES

3.4 commande et surveillance des périphériques

3.5 les imprimantes

3.6 le lecteur de cartes

3.7 performances du SES

Conclusion 219

Annexe 1 : analyse de quelques erreurs de programmation du parallélisme

Annexe 2 : liste des modifications simples à apporter à la machine CII 10070

Bibliographie commentée. 242

INTRODUCTION

The process of preparing programs for a digital computer is especially attractive because it not only can be economically and scientifically rewarding, it can also be an aesthetic experience much like composing poetry and music

D. Knuth. The art of computer programming.

Ce document présente un ensemble de travaux concernant la définition et l'implantation de systèmes informatiques fonctionnant en temps réel et nécessitant un partage de ressources. Ces travaux ont tous été conçus en vue d'applications pratiques. Ils ont toutefois été réalisés avec le souci constant de dégager des principes généraux et des aspects théoriques. Ceux-ci portent sur l'analyse des systèmes, la formalisation de leurs éléments et la méthodologie de réalisation. Car dans l'état actuel de la science informatique, qui passe par une phase de reconnaissance et de classification des objets à étudier, il est important de décrire la réalisation d'expériences particulières en dégageant les structures fondamentales et les idées directrices. [1].*

Une bonne méthode, et peut être la seule valable, à ce jour, pour tester la validité d'un système informatique est de le mettre à la disposition d'utilisateurs, professionnels de l'informatique ou non, et de recueillir leurs opinions. En effet, l'informatique est encore une science expérimentale et l'étude des systèmes n'est pas suffisamment avancée pour qu'on puisse juger une idée ou une organisation sur le plan théorique seul, indépendamment de toute application.

On pourra constater que le premier système présenté est utilisé quotidiennement à bord de navires et que le second a été mis à la disposition d'une communauté de chercheurs.

Nous allons décrire des travaux accomplis, d'une part au centre Coelacanthe, un laboratoire des services scientifiques de la Marine et d'autre part à l'Institut de Recherche d'Informatique et d'Automatique. Tous ces travaux ont été faits en collaboration.

En effet, comme l'esprit humain ne peut aborder qu'un nombre limité de problèmes à la fois, à condition qu'ils ne soient pas trop complexes, et que le rendement d'un programmeur de systèmes est faible, de l'ordre d'une dizaine d'instructions par jour, il en résulte quand on considère la complexité et la taille des problèmes abordés, qu'ils nécessitent toujours un travail d'équipe.

---

* les références renvoient en fin de chaque chapitre.

Les projets réalisés sous notre responsabilité au centre Coelacanthe se décomposent comme suit :

- un sous système de commande numérique de périscope par asservissement échantillonné,
- un sous système d'acquisition, de centralisation et de présentation de données,
- un superviseur de ressources, mettant en place les configurations possibles du système,
- un moniteur de multiprogrammation.

L'ensemble de ces projets reçut le nom de système HALIOTIS*, de l'appellation d'un coquillage marin. Leur réalisation s'est traduite par des programmes d'un total de 117000 mots de 15 bits dont 88 000 instructions, 29 000 mots de données ou de tables de travail.

Les trois premiers projets sont l'application des connaissances d'ingénieur et des techniques d'analyse à des problèmes particuliers. Ils n'ont donné lieu qu'à des notes de travail internes au centre Coelacanthe.

La commande numérique de périscope a soulevé des problèmes qui ont trait au matériel, à l'identification de fonction de transfert avec des non linéarités et en présence de bruit, au calcul de filtres digitaux et à la procédure de dialogue avec l'utilisateur. L'équipe qui a fait le travail comprenait des spécialistes des matériels et de la programmation. Notre contribution personnelle a porté sur l'analyse du problème, le calcul des filtres digitaux et la définition des procédures. La réalisation et la partie expérimentale doivent beaucoup à MM. IMBERT et LUCIANO.

L'acquisition de données, leur centralisation et leur présentation sur écran cathodique de manière à reproduire l'environnement du bâtiment a entrainé l'étude et la spécification de nouveaux matériels ainsi que l'invention de procédures de dialogue très particulières au problème. La réalisation et la mise au point du matériel a été confiée à une équipe extérieure. Notre tâche fut de coordonner

*HALIOTIS - (du grec Halios, marin) : ceux des mers tropicales sont recherchés pour leur coquille ; ceux des régions tempérées, appelés vulgairement "oreille de mer" sont comestibles.

l'ensemble du travail, de faire l'analyse globale du problème et de définir quelques procédures de dialogue et d'entrée-sorties. L'analyse détaillée et la réalisation des programmes furent effectuées par MM. PEIROTES et TELLIER ou dirigées par eux.

La répartition des ressources du système en diverses configurations s'effectue de manière semi-manuelle. Elle a demandé la spécification de permutateurs sur les principales liaisons du système et de complexes panneaux de commande et de supervision. A la répartition des ressources matérielles doit correspondre une adaptation des programmes. Nous avons participé à l'étude du problème et à la définition du superviseur de ressources. Le projet fut réalisé par MM. DERVILLE et LELANN.

La définition des besoins opérationnels et la spécification des matériels de ces trois parties furent l'oeuvre de M. PICOT.

Le quatrième projet relève de l'informatique. Il a consisté à définir l'organisation des programmes de l'ensemble du projet, à écrire les programmes communs (messages d'entrée-sortie en liaison avec les périphériques temps réel, gestion des disques, historique...) et à réaliser un moniteur de multiprogrammation permettant de respecter des contraintes de démarrage et de fin d'exécution des travaux.

Les structures fondamentales des programmes et les idées directrices du moniteur ont fait l'objet d'une publication [2].
Par la suite, il ne sera question que de ce projet et lorsque la dénomination système HALIOTIS sera employée, c'est à lui que la référence sera faite.

Le projet réalisé à l'IRIA se présente d'emblée comme une recherche. Il ne s'agit pas de réaliser un système commercialement exploitable, mais de dégager une méthodologie et un essai de rationalisation dans un domaine où l'empirisme est encore largement prépondérant. Sur le plan technique, on se propose d'apporter une meilleure définition des problèmes soulevés par les systèmes à partage de ressource, de fournir un jeu d'instructions primitives à placer au coeur de toute réalisation et de procurer des mesures sur le compor-

tement des systèmes. [3].

Cette recherche a reçu le nom d'ESOPE, abréviation pour : exploitation simultanée d'un ordinateur et de ses périphériques. Le système ESOPE doit gérer trois sortes de travaux classés ci-après par ordre de priorité décroissante :

- un travail unique en temps réel, caractérisé par l'intervention explicite du temps physique, sous forme d'interruptions d'horloge, à des instants dépendant du problème.
- des travaux conversationnels à partir de terminaux à distance (téléimprimeurs ou consoles alphanumériques).
- un travail de fond, chargé à partir d'un lecteur de cartes ou de bandes. L'enchaînement de ces travaux est automatique, mais se fait en monoprogrammation.

La présence du travail en temps réel est une contrainte importante du système. Ce mode de fonctionnement permet l'interaction de plusieurs terminaux (qui est exclue au cours des travaux en mode conversationnel). Les applications prévues comprennent l'exécution de jeux d'entreprise, de travaux d'enseignement programmé et la simulation de systèmes de gestion en temps réel.

Une autre caractéristique originale du projet est l'utilisation de console graphique. Cette console, fonctionne en même temps que les autres travaux du système. Elle sert à des recherches sur les langages de communication graphique entre l'homme et la machine.

Les travaux en mode conversationnel à partir de téléimprimeurs ou de consoles alphanumériques constituent l'essentiel de l'activité du système. Les utilisateurs de terminaux ont à leur disposition un certain nombre d'outils de programmation conversationnelle réalisés à l'IRIA :

- un sous ensemble du langage PL1, compatible en ce qui concerne la forme externe, avec la version "batch" réalisée par le constructeur de la machine.
- un système de fichiers dont la caractéristique essentielle est la simplicité. Il permet à chaque utilisateur d'accéder à des informations qui lui sont propres et à des informations communes.
- un éditeur de texte.
- un langage LP 10070. Ce langage, fondé sur le PL360 de N. WIRTH est un langage d'assemblage évolué et est

utilisé pour l'écriture et la mise au point du système lui-même.

Le système ESOPE a été réalisé par une équipe qui comprenait MM. BETOURNE, FERRIE, KRAKOWIAK, MOSSIERE et nous même. Le moniteur résident occupe 19000 mots de 32 bits dont 9000 de code et 10000 de données. Il fut l'objet de diverses publications. [4, 5, 6, 7, 8].

Nous présenterons d'abord le système HALIOTIS parce qu'il est le plus ancien et qu'il motiva notre intérêt pour les processus parallèles et parce que les problèmes d'informatique rencontrés sont plus simples. En effet, une seule ressource, l'unité centrale, est partagée. La mémoire est allouée statiquement et le système de fichier est primitif. Nous exposerons en détail l'algorithme d'allocation de l'unité centrale car il tient compte de contraintes sur le temps d'exécution des programmes.

Nous présenterons ensuite le système ESOPE pour lequel les contraintes de temps réel sont moins fortes, mais où les ressources mises en commun et les processus parallèles sont plus nombreux.

Comme la conception d'ESOPE repose sur un petit nombre de concepts que nous avons mis en avant lors de l'analyse du problème et qui ont été suivis tout au long de la réalisation, la seconde partie de ce travail consiste en un exposé du système à réaliser et des mécanismes introduits pour concrétiser les concepts dégagés. Après avoir introduit la notion de processus pour repérer une suite temporelle d'actions et la notion d'usager pour définir un allocataire de ressources, nous montrerons comment la coopération entre les processus peut être assurée à l'aide de sémaphores, nous indiquerons la manière dont nous avons utilisé le matériel disponible pour simuler, par couplage, une très grande mémoire virtuelle et nous exposerons les dangers de blocage et d'écroulement qui se présentent lors de l'allocation des ressources.

Dans une troisième partie, nous montrerons l'utilisation des mécanismes introduits, en insistant plus particulièrement sur la gestion du parallélisme. En effet un travail complémentaire au notre [9] développe les choix qui sont intervenus pour l'allocation des ressources.

Nous présenterons les processus parallèles et leur coopération dans l'allocateur de mémoire et d'unité centrale, dans la communication entre l'utilisateur et le système, et dans la gestion des périphériques.

En conclusion, nous reviendrons sur les concepts de base pour les analyser avec l'expérience de leur utilisation et nous donnerons un aperçu de notre méthodologie de construction de système.

En annexe, le lecteur curieux trouvera la présentation et l'analyse de quelques erreurs de programmation dues au parallélisme des actions, puis la liste de quelques améliorations souhaitables de la machine utilisée pour ESOPE.

Enfin, notre travail se terminera par une bibliographie commentée.

Bibliographie

[1] ARSAC. Les systèmes de conduite des ordinateurs DUNOD 1968

[2] DERVILLE, KAISER, PEIROTES, TELLIER. Le système HALIOTIS . RIRO (nov 1967)

[3] IRIA. Le projet ESOPE. Bulletin de l'IRIA n° 1 (1969)

[4] BETOURNE, BOULENGER, FERRIE, KAISER, KOTT, KRAKOWIAK, MOSSIERE, Process management and resource sharing in the multiacces system ESOPE. Second ACM symposium on operating system principles (oct 1969). CACM 13,12 (dec 1970)

[5] BETOURNE, BOULENGER, FERRIE, KAISER, KRAKOWIAK, MOSSIERE. Présentation générale, notion d'espace virtuel et allocation de ressources dans le système ESOPE. 3 articles présentés au congrès AFCET (sept 1970)

[6] BETOURNE, FERRIE, KAISER, KRAKOWIAK, MOSSIERE. System design using parallel processes. Congres IFIP (août 1971)

[7] BAUDET, FERRIE, KAISER, MOSSIERE. Entrées-sorties dans un système à mémoire virtuelle. Congrès AFCET (nov 1972)

[8] BETOURNE, KRAKOWIAK. Simulation de l'allocation de ressources dans un système conversationnel à mémoire virtuelle paginée. Congrès AFCET (nov 1972)

[9] KRAKOWIAK. Conception et réalisation de système à accès multiple : allocation des ressources. Thèse à paraître.

PREMIERE PARTIE

LE SYSTEME HALIOTIS

Créer n'est pas un jeu quelque peu frivole. Le créateur s'est engagé dans une aventure effrayante qui est d'assumer soi-même jusqu'au bout les périls risqués par ses créatures.

J. Genêt. Journal du voleur

CHAPITRE PREMIER

PRESENTATION DU SYSTEME

Il n'existe pas de méthode rigoureuse permettant de décrire nettement les objectifs à atteindre et de poser clairement les problèmes à résoudre. Aussi le langage courant est-il employé pour la spécification des systèmes.

Une approche entièrement fonctionnelle mêlerait les spécifications des matériels et des programmes. C'est la méthode plus difficile et c'est celle qui demande le plus d'esprit de méthode car le nombre des choix est grand [1].

Une approche pragmatique consiste à décrire en premier les matériels employés. En effet, souvent ceux-ci sont choisis pour des raisons économiques, politiques ou historiques et pas seulement techniques. Ils représentent une contrainte du système. C'est le cas ici.

Pour présenter le système, nous décrirons d'abord le matériel qui nous était imposé, puis les types de fonctions à résoudre et l'organisation générale adoptée. Enfin, à titre d'exemple, nous indiquerons une liste de travaux à exécuter pour la fonction "situation tactique".

1.1 Le matériel [2]

Le système HALIOTIS est organisé autour de 4 calculateurs CAE-133/174. Chacun d'eux comporte :

- une mémoire à tores de ferrite de 32 K mots de 15 bits. Le cycle mémoire est de 2 micro secondes.
- une unité de traitement très simple avec une structure proche d'une machine microprogrammée. Toutefois au lieu d'être dans une mémoire rapide spéciale, les microprogrammes (ou logrammes) sont des sous programmes rangés en mémoire centrale.
- des canaux d'accès à 15 ou 30 bits.
- un seul niveau d'interruption, ce qui conduit à une longue analyse par programme (1 à 2 ms) des causes

d'interruption.

Ce calculateur, conçu vers 1960, ne possède aucun des dispositifs récents suivants :

- protection mémoire,
- registres d'index ou de base,
- mode exécutif,
- horloge temps réel.

Comme on ne peut se passer de cette dernière, elle a été rajoutée dans un périphérique.

Le nerf de ce système est l'ensemble de canaux de communication qui partant du calculateur le raccordent aux périphériques. Ces canaux furent inspirés par le canal UNIVAC introduit à l'occasion des calculateurs AN/USQ-20 du NTDS (Naval Tactical Data System) et sont compatibles avec lui. Le terme canal signifie support logique de transfert d'information ; il n'est pas pris au sens d'organe de raccordement. Ce canal, le premier de ce genre, comporte déjà les principales idées des canaux qu'on trouve sur les calculateurs de la 3° génération. Il est suffisamment simple pour en donner une description complète.

L'une de ses propriétés importantes est l'emploi des mêmes fils pour transmettre les données et les commandes, la distinction entre l'un et l'autre mode étant faite par une ligne spéciale. Le canal se compose des signaux suivants :

- pour le sous canal de sortie :
  - des fils de sortie (30 ou 15 selon les canaux)
  - 1 fil "demande de sortie" DS
  - 1 fil "sortie parée" SP
  - 1 fil "commande externe" CE
- pour le sous canal d'entrée :
  - des fils d'entrée (30 ou 15)
  - 1 fil "demande d'entrée" DE
  - 1 fil "entrée acceptée" EA
  - 1 fil "fin de bloc" FB
  - des fils d'interruption

Le fonctionnement est le suivant :

- le calculateur se connecte sur le canal par une instruction spéciale

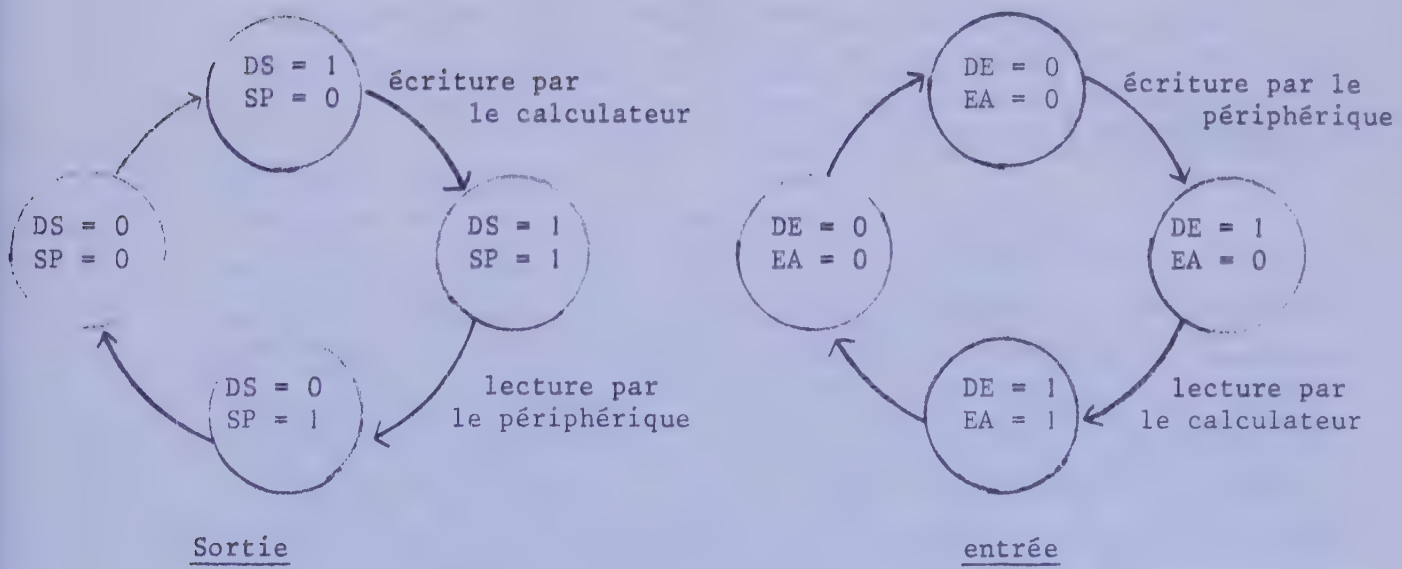
- un des appareils branchés sur le canal est appelé. Pour cela, le calculateur place un code sur les fils de sortie et excite le fil CE. Grâce à celui-ci, tous les appareils reliés au canal sont priés d'examiner le code. L'appareil choisi se connecte alors tandis que tous les autres se déconnectent. Lorsque l'appareil sélectionné est prêt, le transfert peut commencer.

- pour un transfert en sortie, le périphérique met DS à 1. Le calculateur place les données sur les fils de sortie et met SP en 1. Le périphérique peut lire les données pour les exploiter. En fin de lecture, il remet DS en 0. Le calculateur remet SP à zéro et enlève les données. Le cycle peut recommencer dès la disponibilité du périphérique.

C'est le calculateur qui gère l'information à transmettre et qui arrête l'opération quand tous les mots sont sortis.

- pour un transfert en entrée la procédure est identique. Le périphérique place les données sur les fils d'entrée et met DE en 1. Le calculateur lit les données et met EA en 1. Le périphérique remet DE en 0 puis le calculateur remet EA en 0.

La figure ci-après schématise ce fonctionnement :



1.2 Les fonctions

Comme il a été dégagé dans [3], le système comprend 3 classes de fonctions opérationnelles :

- l'acquisition de données,
- la commande d'organes asservis,
- la présentation d'information accompagnée d'un dialogue homme machine et une fonction de gestion (moniteur, superviseur).

Dans les deux premières classes, lorsque l'homme intervient, c'est pour ses qualités mécaniques (filtre ou détecteur d'erreur) ; il est subalterne de la machine.

Dans la 3° classe par contre, la machine est à son service : elle l'informe, il prend les décisions.

L'acquisition de données assure les liaisons entre le système et son environnement. Il s'agit par exemple de coder et de repérer les signaux sonores relevés par les appareils d'écoute du bord (sonores), les signaux électromagnétiques en provenance des radars ou les indications visuelles repérées aux périscopes. Ces opérations sont réalisées avec des degrés d'automatisation divers, le repérage étant en général automatique, et la qualification du signal (identification, valeur, degré de confiance, plage d'incertitude) étant fournie par un opérateur humain.

Le calculateur intervient dans cette fonction pour entrer de l'information dans des tables à une cadence compatible avec la bande passante du phénomène observé. dans le cas présent on peut admettre une seconde entre deux mesures.

La commande d'organes asservis peut agir en amont de la boucle d'asservissement. C'est le cas d'asservissements analogiques qui reçoivent des consignes. Le rôle du calculateur se borne à élaborer ces consignes en fonction des données acquises par ailleurs et des ordres affichés. La commande peut aussi agir dans la boucle d'asservissement. Le calculateur est alors un élément de la boucle de commande. Il doit échantillonner les signaux d'entrée et de sortie, faire le calcul de la compensation digitale et élaborer le signal de commande.

La présentation d'information et le dialogue homme machine a pour but de regrouper les différentes données reçues, de les

corréler et d'en tirer les renseignements synthétiques utilisables (position, vitesse, identification, trajectoire des mobiles). Le calculateur affiche ces données sur divers appareils enregistreurs : table traçante, consoles graphiques avec écran cathodique, pupitres d'affichage de données alphanumériques également sur écran cathodique. Le programme doit calculer l'information de synthèse, préparer et entretenir les diverses images.

Le moniteur doit lancer les divers programmes en réponse aux sollicitations extérieures, assurer la circulation des données et garantir une coopération harmonieuse entre les programmes au cours du temps.

1.3 Organisation du système

Cette organisation s'est élaborée sous l'influence de contraintes provenant soit du problème posé, soit du matériel imposé.

Les données du problème sont intervenues pour diviser le système en deux sous ensembles :

- par prudence, on a voulu doubler les calculateurs et certains matériels vitaux,
- par rentabilité, on a voulu utiliser au mieux ces deux groupes de deux calculateurs. Au lieu d'avoir un seul groupe actif et l'autre en attente, on leur fait réaliser simultanément deux activités. En cas d'avarie dans l'activité vitale, l'autre est arrêtée rapidement et ses calculateurs donnés à la première.

Les fonctions d'une activité peuvent être ordonnées selon trois propriétés :

- priorité : elle définit l'importance d'une fonction.
  Elle est fixée lors de la construction du système.
- urgence : c'est typiquement une propriété de "temps réel".
  Elle définit l'utilité de la fonction et elle exprime que certaines informations perdent toute valeur si elles ne sont pas traitées dans un délai bien déterminé. C'est une fonction décroissante du temps.
- synchronisme : cette propriété traduit le fait que l'exécution d'une partie d'un travail peut être soumise à l'exécution préalable d'une partie d'un autre travail.

Le matériel est intervenu de deux manières :

- pour atteindre la puissance de calcul nécessaire, il a fallu utiliser deux calculateurs, par activité. Chaque groupe de calculateurs comprend un calculateur menant et un mené. Il y a deux groupes dans le système.
- comme la structure du CAE133 ne le permet pas aisément, on n'a pas pu utiliser de programmes réentrants.

En conséquence, quatre niveaux de hiérarchie régissent les programmes :

- le superviseur gère les activités du système, le rôle de ce superviseur est de contrôler les activités en cours et d'assurer, à la demande, leurs chargements en effectuant les mouvements de programmes et les commutations de matériels nécessaires.
- le moniteur gère les travaux constituant une activité du système. Il tient compte de la priorité et de l'urgence des travaux, ainsi que des conditions de synchronisme.
- le travail est une fonction opérationnelle rendant un service déterminé. Un travail est donc un programme défini par ses échanges d'informations avec les autres travaux et les traitements qu'il effectue.
- le module réalise une partie du travail.

Le fractionnement du travail en modules permet de résoudre les conditions de synchronisme car ils possèdent deux propriétés fondamentales :

- l'exécution d'un module peut être soumise à l'exécution préalable d'un module appartenant à un travail différent ;
- l'exécution d'un module étant commencée, celle-ci peut être achevée quel que soit l'état d'avancement des autres travaux.

Pour s'assurer que l'exécution des sous programmes ou des logrammes est bien terminée avant toute nouvelle exécution (sinon ils seraient détruits et le système avec eux), tout module doit s'achever avant de donner le contrôle à un autre travail et toute interruption survenant au cours du module ne peut être traitée que

si elle n'utilise ni logramme ni sous-programme.

L'interruption n'a pas d'effet immédiat sur le déroulement des modules. En particulier contrairement à d'autres systèmes, elles ne conduit pas à un réarrangement des priorités. Sa seule conséquence est de placer un indicateur qui sera examiné à un instant choisi par le programme, en fin de module. On peut donc dire que les commutations d'un travail à l'autre ne sont pas déclenchées par les interruptions, mais seulement par les conditions logiques de synchronisme.

Comme le traitement des interruptions est différé jusqu'à la fin du module en cours, le temps d'exécution de celui-ci ne doit pas dépasser le temps de réponse tolérable pour une interruption. On a pris 50 ms pour ce temps.

A ces niveaux, on a associé des tables de données, accessibles selon la règle des structures de bloc :

bloc du superviseur,
bloc des moniteurs,
bloc des travaux,
bloc des modules.

Ainsi un travail ne peut pas accéder ni aux tables des autres travaux ni aux tables de leurs modules. Il a accès à la table de son moniteur où se trouvent de ce fait les données communes à tous les travaux d'une activité.

Cette hiérarchie d'accès n'existait malheureusement pas dans le langage de programmation employé et il a fallu l'obtenir par une discipline de programmation assez stricte.

1.4 les travaux de la fonction situation tactique

A titre d'exemple, la fonction "situation tactique" chargée de l'acquisition de données, de leur centralisation et de leur présentation sur écrans cathodiques comporte 21 programmes-travaux.

1 - remplissage des temps morts, servant seulement à rendre stable le mécanisme de sélection des travaux par le moniteur (travail périodique) . Le moniteur est adapté à un certain régime permanent - il faut le stabiliser dans les initialisations ou

certains transitoires.

2 - lecture des actions. Tout message ou toute demande de service est validée par une touche spéciale, dite touche "action". Ce travail lit périodiquement les touches "action" pour détecter les demandes et les remettre à zéro (travail périodique).

3 - transfert par canaux A et D. Ce travail gère ces canaux. Le canal A sert de liaison entre deux calculateurs, le canal D est un canal simultané. Ce travail est demandé soit par une interruption de fin de transfert sur le canal D, soit par toute demande de transfert de la part d'un autre travail (travail apériodique).

4 - remplissage du tampon Table traçante. Il place dans le tampon de la table traçante les mots nécessaires à la visualisation d'un plot (travail périodique).

5 - présentation de la situation sur l'écran de la console. Il compose le tampon de la console (travail périodique).

6 - gestion des messages à destination du tote console. Il examine la liste des messages à destination du tote de la console et lorsque la liste n'est pas vide, appelle le travail n° 18 qui compose effectivement le message (travail périodique).

7 - gestion des messages à destination du tote pupitre. Il examine la liste des messages à destination du tote du pupitre de détection sous marine et lorsque la liste n'est pas vide, appelle le travail n° 18 (travail périodique).

8 - traitement de l'action GCO2-D. Lorsque le travail n° 2 a lu une "action" sur le GCO2-D (appareil d'écoute microphonique), il active ce travail qui analyse le message qui l'accompagne et effectue les traitements correspondants (travail apériodique).

9 - commande et lecture du GCO2-E. Le GCO2-E est un appareil d'écoute automatique. Ce travail réalise le calcul et l'envoi du gisement des bruiteurs que l'on désire surveiller. Il lit et traite le résultat des recherches. (travail périodique).

10 - traitement de l'action DUUX. Ce travail est appelé par le travail n° 2 lorsqu'une "action" DUUX (télémètre acoustique) est présente. Il analyse le message et effectue les traitements appropriés. (travail apériodique).

11 - traitement de l'action Sonar. Il est appelé par le travail n° 2 lorsqu'une "action" Sonar se présente (travail apériodique).

12 - traitement de l'action 1 pupitre. Il est appelé par le travail n° 2 pour traiter le message qu'un opérateur a composé sur le pupitre de détection sous marine (travail apériodique).

13 - traitement de l'action 1 console - analogue au précédent, mais pour la console. (travail apériodique).

14 - liaison des programmes tactiques avec le programme de recalage astral. Ce travail permet de transférer du calculateur menant au calculateur mené des données globales élaborées par celui là (travail périodique).

15 - contrôle opérationnel n° 1. Il effectue un contrôle sommaire du calculateur, des unités de liaison, de l'horloge et dépose des messages de tests destinés aux périphériques tactiques (console et pupitre, table traçante et multiplex de codage) (travail apériodique).

16 - contrôle opérationnel n° 2 - Appelé par le travail précédent, il traite les réponses des périphériques tactiques aux messages de test. (travail apériodique).

17 - traitement de l'action ARUR. Il exploite les messages en provenance de l'ARUR, appareil détecteur de radar. (travail apériodique).

18 - composition des messages à destination des totes. Ce travail est appelé par l'un des travaux n° 6 ou n° 7 prépare les messages à visualiser sur les totes. (travail apériodique).

19 - constitution des historiques. Il range sur un tourne disque réservé, les données qui constituent l'essentiel des informations sur la situation actuelle autour du bâtiment (travail périodique).

20 - liaison avec l'engagement tactique. Même rôle que le travail n° 14, mais pour d'autres données. (travail périodique.)

21 - liaison avec l'azimétrie. Il assure dans un troisième cas, la transmission de données entre calculateur mené et calculateur menant. Cette fois la liaison s'établit dans les deux sens (travail périodique).

1.5 Bibliographie

[1] CORBATO. Sensitive issues in the design of multiuse systems, MO 105, MIT (dec 1968)

[2] CAE. Manuel de référence de l'unité centrale CAE 133, note technique CII 1493

[3] DERVILLE, KAISER, PEIROTES, TELLIER. Le système HALIOTIS. RIRO 1ère année n° 6 (nov 1967).

CHAPITRE 2

LES PROBLEMES FONDAMENTAUX

Nous analysons dans ce chapitre, deux problèmes importants des systèmes en temps réel, la gestion du parallélisme des travaux et l'allocation des ressources.

2.1 Gestion du parallélisme des travaux

Les travaux d'HALIOTIS évoluent parallèlement au cours du temps et coopérent entre eux au fur et à mesure de leur évolution. Le mécanisme utilisé pour régir cette coopération est un mécanisme direct, en ce sens qu'un travail agit directement sur les variables d'état d'un autre travail.

La coopération entre les travaux parallèles du système dépend des conditions de synchronisme : l'exécution d'un travail dépend de l'exécution préalable de modules appartenant à des travaux différents ou de la réalisation de certains événements (fin d'une entrée-sortie, par exemple).

Tout travail produisant un événement significatif le signale aux travaux qui sont concernés. Pour cela dans le moniteur figure par travail, un vecteur "autorisation d'exécution du travail". Celui-ci indique, pour chaque événement qui, à un moment ou à un autre, a un effet sur l'exécution de ce travail, si il s'est produit ou non. C'est donc ce vecteur qui est modifié par le travail ayant produit l'événement significatif.

Par ailleurs, le sous ensemble des événements attendus à un instant donné, sous ensemble qui exprime les conditions actuelles du synchronisme pour un travail, est indiqué dans un mot "masque du travail". Le travail ne peut se poursuivre que lorsque tous les événements appartenant à ce sous ensemble se sont réalisés.

Remarquons que cette méthode permet aisément de faire bloquer indéfiniment un travail par le moniteur. Il suffit d'introduire dans le sous ensemble des événements attendus un événement produit par

le moniteur. En validant ou invalidant cet événement pour chaque travail, le moniteur autorise ou supprime l'exécution de celui-ci. Ce mécanisme est utilisé pour stabiliser le système. En cas de surcharge, les travaux les moins importants sont arrêtés et retardés jusqu'à ce que la charge redevienne normale.

Cette méthode est valable pour un système où le nombre des travaux est fixe, ainsi que les événements qui les font évoluer. Si il y a modification du nombre des travaux et des événements, il faut reprogrammer avec soin toutes les interactions.

Dans un système très spécialisé et peu évolutif comme l'est HALIOTIS, cette méthode s'est révélée suffisante. L'existence d'un vecteur "autorisation d'exécution" et d'un mot "masque" par travail s'est montrée au contraire très utile pour suivre l'évolution du système, que ce soit pendant sa mise au point ou durant son fonctionnement opérationnel.

2.2 Répartition des ressourses

Les ressources d'HALIOTIS sont constituées de périphériques opérationnels, de disques à tête mobile, de la mémoire principale et de l'unité centrale. A part l'unité centrale, les ressources sont distribuées très simplement.

2.2.1 Ressources autres que l'unité centrale.

Les périphériques opérationnels sont attribués de manière fixe à des travaux précis. Ainsi le travail de transfert par canaux A et D reçoit tous les périphériques branchés sur le canal D. Aucun transfert avec ces périphériques ne peut être effectué par un autre travail.

Les disques à tête mobile ont eux aussi reçu des affectations fixes. L'un d'eux est réservé au travail de constitution des historiques qui le gère comme un tampon circulaire. Les autres sont attribués au moniteur qui y trouve les programmes des différentes configurations opérationnelles.

La mémoire est divisée en deux parties. Dans la première, les programmes résidents sont chargés lors de la mise en place de la configuration et y demeurent durant toute la durée de celle-ci. La seconde partie de la mémoire sert de zone de recouvrement pour les

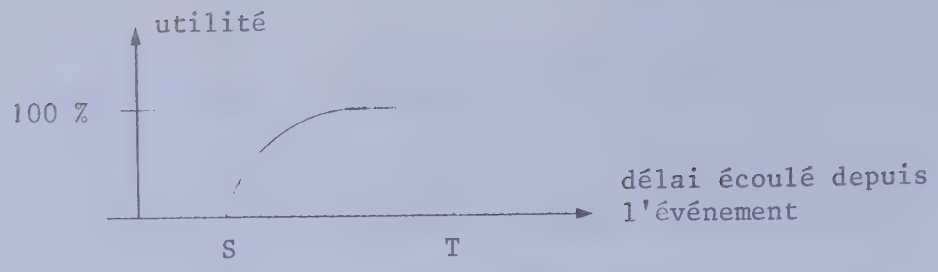
autres programmes qui sont rangés sur disque. La gestion de ces programmes est simple : ils ont tous la même adresse d'origine, celle de la deuxième partie de la mémoire et ont une taille maximale fixée à 2720 mots. Les variables globales qu'ils manipulent, variables de synthèse, état des pistes, etc..., sont résidentes. Il en résulte, puisqu'un module n'est pas interrompu, que ceux-ci n'ont pas besoin d'être réécrits sur disque. Le transfert ne se fait que dans un sens, en lecture.

2.2.4 Unité centrale

2.2.41 Contraintes

L'attribution de l'unité centrale est une opération complexe. En effet, on doit tenir compte des particularités du temps réel : la réponse à certains stimuli ou événements extérieurs au système informatique doit survenir avant un délai fixé. Passé celui-ci, l'élaboration de cette réponse devient parfaitement inutile, parfois nuisible. Ainsi par exemple, la commande d'appareils par la technique des asservissements échantillonnés suppose un échantillonnage régulier. On sait calculer des filtres numériques qui tiennent compte de retards dans la connaissance des signaux ou dans l'envoi des ordres pourvu que ces retards soient constants ou qu'ils possèdent des propriétés statistiques données. Au delà d'un certain seuil, le déphasage introduit par le retard est tel que l'asservissement devient instable. Plutôt que de soumettre la partie mécanique de l'asservissement aux à-coups qui en résulteraient, il vaut mieux sauter l'envoi d'un ordre et laisser l'asservissement en boucle ouverte la durée de deux échantillons. C'est ce qui est fait dans HALIOTIS.

On peut traduire cette propriété des problèmes "temps réel" par la courbe d'utilité d'un programme en fonction du délai écoulé depuis l'arrivée de l'événement significatif.



Au-delà du seuil T, le programme devient inutile. Dans le cas d'un travail périodique, il intervient un second seuil S en deça duquel l'exécution du programme n'a pas de sens.

L'algorithme d'attribution de l'unité centrale tient compte de cette propriété des programmes, d'une priorité fixée lors de la programmation du système et des conditions de synchronisme.

Cet algorithme assure encore deux servitudes :

- maintenir l'activité du système telle que le seuil T ne soit dépassé pour aucun programme.
- commander le chargement de programmes dans la zone de recouvrement.

2.2.42 Variables utilisées

La durée d'exécution des travaux n'est pas prévisible : aussi ne peut-on établir de calendrier pour le début et la fin de chaque travail. Lorsqu'il est appelé, le moniteur doit choisir le travail auquel donner l'unité centrale : ce travail est par définition le travail actif.

L'appel du moniteur se fait à des instants privilégiés :

- lorsque le travail actif se termine,
- lorsque l'urgence d'un travail non actif devient telle qu'il doit être activé pour respecter les contraintes du temps réel.

Pour faire son choix, le moniteur utilise le modèle suivant de l'évolution d'un travail :

- à la date D, il est demandé
- à la date C, il peut commencer
- à la date F, il faut qu'il ait commencé
- à la date T, il doit être terminé

Sur l'axe du temps on a : $D \leqslant C \leqslant F < T$

Pour chaque travail, on conserve donc les grandeurs :

- DP1 = C-D, temps d'attente minimum du travail,
- DP2 = F-C, fourchette d'initialisation du travail,
- DTM = T-F, durée maximum du travail.

Pour un travail apériodique, la date D est celle de l'événement qui en demande la réalisation. Lorsqu'il s'agit d'un travail périodique, c'est le moniteur qui relance la demande de travail, en fonction de sa période P, qui remplace alors DP1.

Pour bien utiliser l'organisation en module, la grandeur DTM a été remplacée par une suite de grandeurs dt associées à chaque module et donnant leur limite de début d'exécution. Bien entendu

$\sum dt_i \leqslant$ DTM avec i = 1,2,.../k ; k étant le dernier module effectué.

Appelons I l'instant où commence l'exécution du premier module d'un travail. Si le système fonctionne bien, on a :

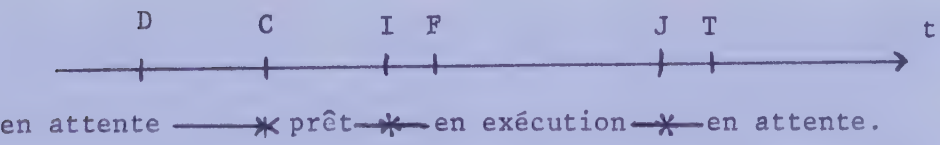
$$C \leqslant I \leqslant F$$

Soit J l'instant où ce travail se termine. En cas de bon fonctionnement,

$$I < J \leqslant T.$$

Le moniteur considère trois états pour un travail :

- avant l'instant C et après J, il est en attente
- de C à I, il est prêt
- de I à J, il est en exécution



Comme il a été vu dans la gestion du parallélisme, un travail en exécution peut se trouver bloqué momentanément en attente d'événement.
Pour chaque travail, le moniteur entretient les variables suivantes :

- état du travail
- DAT1 : date de passage à l'état prêt.
  - Pour un travail apériodique, DAT1 vaut l'infini tant que la date D n'est pas atteinte. Dès lors, il vaut C. A l'instant I, il vaut à nouveau l'infini.
  - Si le travail est périodique, DAT1 est entretenu. A chaque instant C, il devient DAT1 : = DAT1 + P, P étant la période du travail.
- DAT2 : date limite à considérer.
  - Tant que le travail est en attente, DAT2 vaut l'infini.
  - Quand le travail passe prêt, DAT2 vaut F : DAT2:=DAT1+DP2.
  - Lorsque le travail est en exécution, DAT2 est modifié à la fin de chaque module et donne la date limite pour l'exécution du module suivant.
    DAT2:=DAT2+dt_i.
  - quand le travail se termine. DAT2 est remis à la valeur infinie.
- urgence DT = DAT2 - H où H est l'heure courante.

Deux autres grandeurs, fixées à la constitution du travail sont utilisées par le moniteur :

- la priorité π
- le rang de suppression R. C'est le rang d'élimination du travail en cas de saturation du système. A un instant donné, il existe un rang de suppression critique RO attaché au moniteur. Un travail ne pourra être exécuté que si :

$$R > RO$$

2.4.43 Fonctionnement

On peut maintenant décrire sommairement le fonctionnement du moniteur. Celui-ci doit :

a) séparer les travaux en deux classes, ceux qui ne demandent pas l'unité centrale parce qu'ils sont en attente et ceux qui la réclament. Cette séparation est indiquée par l'état du travail. La première fonction du moniteur est de faire passer à l'état prêt, les travaux en attente dont le DAT1 a dépassé l'heure courante H.

b) choisir un travail parmi ceux qui demandent l'unité centrale et qui ne sont pas bloqués en attente d'événement. La fonction de choix utilisée est :

$$C = \pi + \alpha \, DT \quad (\alpha = \text{constante, actuellement } \frac{1}{32})$$

Le moniteur choisit pour exécution le travail ayant la fonction de choix minimmum.

c) veiller au maintien des conditions du temps réel.

- si l'urgence DT d'un travail devient négative, c'est un signe de saturation du système : il faut retirer des travaux. Pour éviter des fluctuations, on se donne un certain seuil pour DT (seuil négatif) en deça duquel le moniteur augmente le rang de suppression critique RO.

- on fait une estimation de la charge du système à l'aide d'une fonction de charge F calculée au moyen de la relation

$$F \text{ nouveau} = \beta . F \text{ ancien} + (1-\beta) . DT \text{ min}$$

DT min est la plus petite urgence des travaux, à l'instant considéré. La constante β vaut actuellement 0,75.

Si F dépasse un certain seuil S, le moniteur diminue le rang de suppression critique RO.

- Ayant fait varier le seuil de suppression critique, le moniteur supprime ou réintègre les travaux en conséquence.

d) appeler en mémoire le travail à exécuter. On a vu que les travaux étaient divisés en deux classes, ceux qui sont résidents et ceux qui ne le sont pas. Pour lire sur le disque un travail non résident, il faut exécuter les opérations suivantes : déplacement de la tête de lecture du disque pour atteindre le cylindre sur lequel le programme est enregistré, et lecture du cylindre lui-même. Ces deux opérations prennent un temps considérable, mais seule la plus courte, la lecture, immobilise l'unité centrale. Le déplacement de la tête peut être mené en parallèle avec l'exécution d'un programme.
Pour profiter de cette simultanéité le moniteur considère deux travaux celui qu'il vient de choisir, Tc, et celui qu'il a choisi lors du passage précédent, Te. Lors de ce passage, il a, si Te n'est pas résident, commandé le déplacement de la tête de lecture. Au passage actuel, il n'a plus qu'à lire le texte de Te et, si nécessaire commander le déplacement de la tête pour Tc, avant d'activer le travail Te.

e) se protéger contre la monopolisation de l'unité centrale par un travail. Pour cela, il existe une horloge temps réel appelée sablier. Ce sablier est décompté toutes les millisecondes et envoie une interruption quand il est vide, ce qui permet au moniteur de reprendre le contrôle dès la fin du module en cours. L'urgence DT la plus petite, celle du travail choisi Tc exceptée, détermine le répit avant un dépassement possible du DAT2 la plus proche par l'heure courante. La valeur de ce répit est placée dans le sablier.

Lorsque le sablier est vide, il est immédiatement rechargé avec la valeur de la durée maximum d'exécution d'un module de manière à permettre au module en cours de se terminer normalement. La réception de deux interruptions de sablier consécutives est alors un signe de boucle indéfinie dans un module ou de panne dans l'unité de calcul.

La programmation du moniteur a été faite selon le schéma suivant :

```
Procédure moniteur ;
          begin
          comment : initialisations diverses, et entre autres : ;
          Te : = Tc ;
          H : = heure lue à l'horloge, en millisecondes ;
          CMIN : = ∞ ; ROO : = RO ; DTO : = DT1 : = ∞ ;
A :       Comment : balayage de la table des travaux de dimension N ;
          for n : = 1 step 1 until N do
               begin
               if (état[n]= attente) & (H ≥ DAT1[n]) then
                    begin
                    état[n]: = prêt ;
                    DAT2[n]: = DAT1[n]+ DP2 ;
                    if P = ∞ then
                         begin comment : travail apériodique ; DAT1 : = ∞ end
                    else
                         begin comment : travail périodique ; DAT1 : = DAT1 + P end
                    end ;
```

```
        comment : calcul de l'urgence et augmentation éventuelle du rang
                  de suppression ;
        DT[n] : = DAT2[n]- H ;
        if DT[n]< - tmem then RO : = RO + 1 ;
        comment : tmem est le temps moyen d'exécution d'un module ;
        comment : détermination des deux urgences les plus petites parmi
                  les travaux déjà examinés. Soit DTO et DT1 ces deux
                  urgences ;
        if DT[n]< DTO then
             begin DT1 : = DTO ; DTO : = DT[n]end
        else
             if DT[n]< DT1 then DT1 : = DT[n];
        comment : calcul de la fonction de choix
        if état[n]≠ attente then
             begin
             C[n]: = Π[n]+α* DT[n];
             if C[n]< CMIN then
                  begin
                  CMIN : = C[n];
                  Tc : = n
                  end
             end
        end

B :     comment : calcul de la charge du calculateur ;
        F : = β * F + (1-β) * DTO ;
        if F > seuil then RO : = RO - 1 ;
        comment : suppression ou réintégration des travaux ;
        if RO < ROO then réintégrer tous les travaux dont le rang de suppression
             R est plus grand que RO ;
        if RO > ROO then supprimer les travaux dont le rang de suppression R
             est plus petit que RO ;

C :     comment : appel en mémoire du travail à exécuter ;
        if Te non résident then
             if tête du disque en place then appel de Te en mémoire else goto C ;
        if Tc non résident then lancer le déplacement de la tête du disque ;
```

```
D :     comment : chargement du sablier ;
        DT2 : = if DTO ≠ DT [Tc] then DTO else DT1 ;
        chargement du sablier avec DT2 ;
        end ;
```

Un autre module important est celui du traitement des interruptions.

Procédure action première d'interruption :

```
        begin
        boolean   ITDIFFEREE, ITSABLIER ;
        ITSABLIER : = FALSE ;
        if interruption sablier then
             if ¬ ITSABLIER     then
                  begin
                  charger le sablier avec la valeur maximum d'exécution
                  d'un module ;
                  ITSABLIER : = TRUE ;
                  end
             else erreur de programmation du système ou erreur machine ;
        ITDIFFEREE : = TRUE ;
        end ;
```

La variable booleenne ITDIFFEREE est testée à la fin de chaque module d'un travail et lorsqu'elle est TRUE, le module appelle un module de traitement d'interruption différée qui remet ITDIFFEREE à FALSE et fait l'action correspondant à l'interruption. Cette action peut aboutir à appeler le moniteur. C'est en particulier le cas si ITSABLIER = TRUE. Au cours de son action, le moniteur réinitialisera ITSABLIER à la valeur FALSE.

DEUXIEME PARTIE

MECANISMES DE BASE DU SYSTEME ESOPE

On fait la science avec des faits, comme on fait une maison avec des pierres ; mais une accumulation de faits n'est pas plus une science qu'un tas de pierres n'est une maison.

H. Poincaré. La Science et l'Hypothèse

CHAPITRE PREMIER

PRESENTATION DU SYSTEME

Nous abordons maintenant la présentation du système ESOPE, qui sur le plan de l'informatique représente une étape nouvelle par rapport à HALIOTIS. Dans ce dernier intervenaient deux problèmes essentiels, la gestion du parallélisme des travaux et l'allocation de l'unité centrale suivant des critères particuliers au temps réel. Dans ESOPE, nous avons essayé de traiter, à l'occasion d'un petit système, tous les problèmes qui se posent lors de la réalisation d'un système de "time sharing" le plus général. Aussi non seulement nous retrouvons la gestion du parallélisme des processus et l'allocation de l'unité centrale, mais il s'y ajoute aussi des problèmes de gestion de la mémoire virtuelle, de conservation d'information dans les fichiers en relation avec la mémoire virtuelle, d'allocation conjointe de la mémoire et de l'unité centrale, de communication entre un client conversationnel et le système et d'intégration des périphériques non conversationnels.

Ici encore nous adopterons la démarche consistant à présenter le matériel en premier parce qu'il nous était imposé comme contrainte. Puis nous indiquerons les fonctions de base à remplir et nous préciserons les concepts sur lesquels reposent notre construction du système.

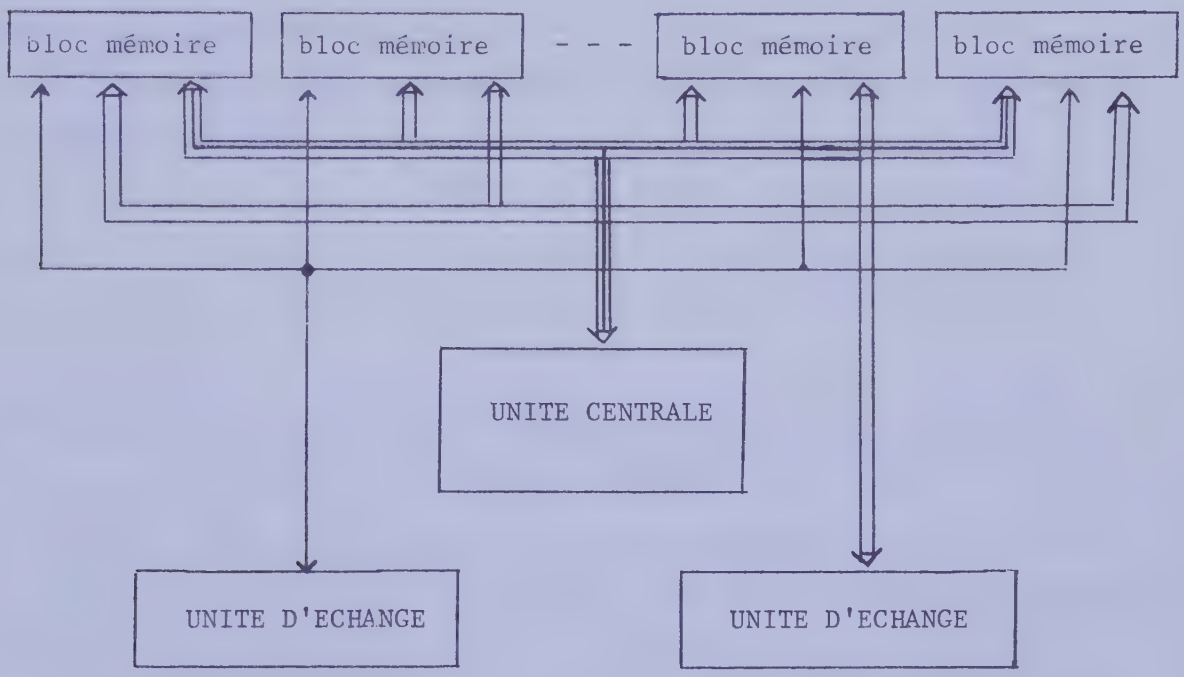
1.1 Le matériel

L'élément de base du système est le calculateur CII 10070, version française de la machine SDS SIGMA 7. [1].

Sa structure comporte trois éléments principaux :

- un ensemble de 8 blocs de mémoire, chaque bloc ayant sa propre circuiterie d'accès et étant connecté à 3 "bus" d'information.
- une unité centrale reliée à un des bus

- deux unités d'échange, chacune étant reliée à un bus. Chaque unité d'échange assure la liaison entre la mémoire et les organes périphériques, via une unité de liaison par périphérique.



La mémoire est à tores de ferrite de 128 K mots de 32 bits (16 K mots par bloc). Le cycle de mémoire est d'environ 850 nano secondes. A chaque bloc est associé un verrou d'écriture qui permet une protection mémoire. L'un des blocs, associé aux adresses hautes de la mémoire a reçu un circuit d'accès spécial qui permet d'ajouter 3 accès supplémentaires. L'un de ces accès est utilisé pour relier la console graphique directement à la mémoire.

L'unité centrale comprend divers dispositifs conçus pour le temps réel et le partage de ressources :

- une pagination de l'adressage virtuel généré par l'instruction. La correspondance entre la page virtuelle et la page physique est réalisée par une mémoire topographique très rapide qui est adressée avec le numéro de page virtuelle. La mémoire virtuelle, de 128 K mots, est formée de 256 pages de 512 mots.
- un dispositif de protection associé à chaque page virtuelle avec les options suivantes :

- tous accès
- accès en lecture seulement
- écriture interdite
- aucun accès

- trois jeux de 16 registres commutables. A un instant donné un programme ne peut travailler qu'avec un seul jeu, mais on peut aussi dans le cas de plusieurs programmes coexistant sur la machine associer un jeu de registre par programme.
- le regroupement du contexte d'un programme dans un double mot d'état programme : DMEP (en anglais : program status double word, ou PSD).
- un mode exécutif qui seul a le droit d'exécuter certaines instructions privilégiées, dont la modification du DMEP.
- un dispositif très élaboré d'interruption de programme et de trappe d'instruction. Des instructions particulières (instructions CAL) utilisent ce mécanisme pour permettre un appel au superviseur.
- quatre horloges, compteurs de temps réel. Elles peuvent être incrémentées à des fréquences variables, la plus haute étant 8 kHz.

Les unités d'échange assurent les entrées-sorties en simultanéité avec l'unité centrale. Elles communiquent avec l'unité centrale par une liaison directe qui permet à cette dernière de leur envoyer des ordres de départ, d'arrêt et de test. Elles trouvent en mémoire un programme qui indique l'enchaînement de leurs actions. Elles sont reliées aux coupleurs de périphériques par un câblage unique appelé interface standart. Cet interface véhicule des données, des adresses de périphérique et des ordres et il permet de mélanger des données en provenance ou à destination de périphériques divers.

On trouvera sur la planche I* un schéma de la configuration matérielle du système ESOPE.

Elle comporte :

- deux unités d'échanges multiples à 400 000 octets/s
- un ensemble de 3 RAD, disques à têtes fixe, chacun d'eux contenant 3 millions d'octets, ayant un temps de révolution de 34 millisecondes et un débit de 175 000 octets/s (une page est transférée en 12 millisecondes)
- des périphériques lents classiques dont 6 dérouleurs de bande magnétique à 60 000 octets/s

* les planches figurent en fin de chaque chapitre.

- une mémoire de masse de 180 millions d'octets, constituée de disques à têtes mobiles, et appelée DIMAS
- 16 machines à écrire SAGEM 8/200 à 15 caractères/seconde et fonctionnant en "full-duplex"
- 8 consoles alphanumériques SYNECRAN (INFOTON) comportant un écran cathodique et ayant pour l'unité centrale un comportement identique à celui des SAGEM 8/200.
- 1 console graphique SINTRA VU 2000, en connexion directe à la mémoire du 10070.

Ce matériel comporte quelques contraintes pour la programmation :

- il n'y a pas de dispositif câblé de translation dans la mémoire virtuelle. Un exemple de tel dispositif est constitué par les registres de base. Les registres d'index du 10070 sont déjà employés pour accéder aux octets et aux demi-mots. Aussi ne peut-on pas avoir aisément de code réentrant dans une même mémoire virtuelle.

- il n'y a pas d'indicateur d'utilisation d'une page de mémoire (du type "Least recently used" comme dans la machine IBM 360/67), ni d'indicateur d'écriture dans une page. Il faut les simuler en employant le dispositif de protection en mémoire virtuelle.
- les disques RAD ont un débit très faible.

1.2 Les fonctions

La principale destination du système concerne le dialogue entre l'homme et des programmes chargés dans la machine. Les organes de dialogue sont les téléimprimeurs SAGEM, les SYNECRAN et la console graphique.

Une autre destination, subalterne par construction, se rapporte à l'exécution non conversationnelle. Dans ce cas, l'information circule entre des organes périphériques classiques, bande magnétique, cartes perforées, disques magnétiques amovibles, et le calculateur.

La notion de priorité, associée à un niveau d'interruption, notion que l'on rencontre dans les systèmes en temps réel, ne s'applique guère. Si 3 niveaux apparaissent :

- travail interactif prioritaire,(travail dit en temps réel)
- autres travaux conversationnels,
- travail de fond,

en fait le moniteur doit être essentiellement conçu pour gérer des travaux conversationnels qui sont tous sur un même plan.

Ceci explique que les niveaux d'interruption et la notion de priorité qui y est associée n'aient pas été employés extensivement. Une autre raison de ce fait est que l'interruption est un outil de gestion de l'unité centrale qui emploie la structure de pile. En effet, le mot état programme DMEP du programme interrompu est garé dans une mémoire dépendant de l'interruption, créant ainsi une pile de programmes dans le cas d'interruptions successives. Or dans ESOPE, comme on le verra plus loin la gestion de la mémoire précède la gestion de l'unité centrale et la structure de pile n'est pas employée. Il devenait donc illogique d'utiliser les interruptions pour cela.

L'analyse des systèmes à partage de ressource a fait apparaître trois fonctions qui doivent être remplies dans tout système :

- la communication entre des travaux qui s'exécutent en parallèle et qui coopèrent entre eux en partageant des données ou des ressources
- la définition d'une mémoire virtuelle et son extension par un système de fichiers permanents
- l'attribution des ressources aux travaux qui se les partagent.

Il est commode de séparer les problèmes rencontrés en deux classes :

- ceux qui sont intrinsèques ou inhérents au problème lui-même
- ceux qui sont technologiques et qui dépendent des limitations technologiques rencontrées à un instant donné.

Les principaux problèmes intrinsèques concernent la communication entre des travaux parallèles, l'accès à l'information, le partage de celle-ci entre plusieurs utilisateurs et sa protection contre les erreurs du système ou des utilisateurs.

Les problèmes technologiques sont traités dans la distribution des ressources en nombre limité.

1.3 Les principes

La conception d'ESOPE repose sur un petit nombre de principes qui ont été dégagés à la suite d'une étude logique des problèmes et qui ont été suivis tout au long de la réalisation du système. Les compromis, rendus parfois nécessaires pour tenir compte des contingences du matériel ont respecté ces principes.

Le premier concept introduit est celui de processus séquentiel.

On peut distinguer plus ou moins arbitrairement dans un système un certain nombre fini de parties. Le choix de ce partage dépend de la commodité d'ordonnancement des parties au cours du temps. Ce partage définit ainsi un nombre fini de fonctions ou de tâches. On appelle processus la mise en oeuvre d'une tâche. La notion de processus est dynamique. Elle implique un déroulement dans le temps, une succession d'actions.

A la notion de processus sont attachés les éléments suivants :

- algorithme : description complète et non ambigüe de la tâche à accomplir
- données : informations sur lesquelles opère le processus
- ressources : moyens nécessaires à l'exécution d'un processus.

On distingue deux classes de ressources. Les ressources passives conservent l'information sans la modifier. Ce sont les mémoires du système. Les ressources actives ou processeurs exécutent des algorithmes, c'est-à-dire les interprètent pour exécuter des actions.

Un processus séquentiel ne dispose, à un instant donné que d'un processeur au plus. Quand la réalisation d'une tâche demande l'emploi simultané de plusieurs processeurs, il est introduit autant de processus que de processeurs pouvant fonctionner simultanément.

Il est important de bien saisir la différence entre la notion de processus et celles d'algorithme, de programme ou de processeur :

- plusieurs processus peuvent correspondre au même algorithme (ou programme)
- un processus peut être amené à des instants successifs à utiliser

des processeurs différents.

A un instant donné un processus se trouve dans l'un des deux états suivants :

actif : il progresse dans le déroulement de son algorithme, ou bien il progresserait s'il avait toutes les ressources physiques dont il a besoin,

bloqué : le déroulement de son algorithme est arrêté et, pour repartir, il attend un signal d'un autre processus. C'est dont un arrêt pour raisons intrinsèques.

A tout moment, il existe un ensemble minimal d'information dit vecteur d'état du processus qui définit complètement l'évolution du processus. Si le processus est interrompu à un instant quelconque et l'état de ses ressources modifié de façon quelconque, la donnée du vecteur d'état à l'instant de l'interruption permet de réactiver le processus à un instant ultérieur comme si l'interruption n'avait pas eu lieu.

Ainsi le vecteur d'état d'un processus qui s'exécute sur une unité centrale se compose

- du mot d'état de programme de l'unité centrale
- du contenu des registres généraux
- du contenu des mémoires accessibles par le processus.

Pour distinguer, à propos des processus les problèmes technologiques et les problèmes intrinsèques, il est utile d'introduire la notion de machine virtuelle, ensemble de ressources virtuelles dont un processus a besoin pour s'exécuter. On ne tient alors plus compte des conflits qui proviennent de la limitation des ressources et les problèmes intrinsèques sont réglés à l'aide de machines virtuelles. Par ailleurs, un ensemble de processus privilégiés du système, appelés processus d'allocation, règlent les problèmes technologiques d'allocation des ressources.

Cette démarche a déjà été employée pour définir les états d'un processus puisque dans l'état actif un processus peut être arrêté pour des raisons technologiques. On introduit souvent pour traiter ce cas un état supplémentaire, l'état prêt. Un processus prêt serait actif s'il avait toutes ses ressources. On parle quelquefois aussi de blocage technologique par opposition au blocage intrinsèque.

Le second concept important est celui d'usager.

L'analyse du problème montre qu'il existe dans tout système d'accès multiple des sous ensembles de processus tels que ceux-ci communiquent entre eux à l'intérieur d'un sous ensemble et n'ont aucune relation intrinsèque (c'est-à-dire figurant dans leur algorithme) avec les processus d'autres sous ensembles. Ces sous ensembles forment une partition de l'ensemble des processus, hormis les processus technologiques de l'allocateur. On appelle alors usager un tel sous ensemble. Par extension les processus du moniteur chargés de résoudre les problèmes technologiques forment un usager particulier, l'usager moniteur.

Il est alors commode d'utiliser cette notion d'usager dans la définition de la mémoire virtuelle et dans l'attribution des ressources. A chaque ensemble de processus formant un usager, on associe une mémoire virtuelle, telle que le fournit le matériel. Dans le cas du 10070, c'est une mémoire formée d'un seul segment de 128 K mots de 32 bits. Tous les processus d'un usager se partagent ce segment et communiquent entre eux par adressage à l'intérieur du segment. De même l'usager, et non le processus, est l'élément auquel sont distribuées les ressources. Ainsi les périphériques lents tels que lecteur de carte, imprimantes et bandes magnétiques sont attribués en permanence à un usager privilégié, l'usager PTT.

Chaque machine à écrire ou chaque console est attribuée à un usager dès qu'un client du système s'y présente pour une session. L'usager PTT reçoit encore un statut privilégié puisqu'il est en session permanente et qu'en conséquence il reçoit, à demeure, le télétype de l'opérateur. Quant aux ressources distribuées dynamiquement, mémoire et temps d'unité centrale, elles sont partagées entre tous les usagers.

En résumé, les différents usagers du système sont les suivants :

- le moniteur, seul usager travaillant en mode maître et en mémoire physique. De la mémoire lui est attribuée en permanence dès la construction du système. Il est prioritaire pour l'emploi du temps de l'unité centrale. Les disques RAD lui sont réservés. Il comprend

6 processus chargés d'allouer le restant de mémoire et de temps d'unité centrale aux autres usagers, 1 processus de communication avec l'opérateur, 1 processus de surveillance des périphériques et autant de processus pilotes de périphériques qu'il y a de contrôleur de périphériques

- un usager par terminal conversationnel (machine à écrire SYNECRAN ou console graphique), travaillant en mode asservi et en mémoire virtuelle. Il comprend au moins deux processus, le processus premier qui est la partie du système travaillant en virtuel et dont le code est réentrant et le processus final. En général, cet usager comprend un autre processus qui utilise le code d'un traducteur (éditeur de texte, PL1 conversationnel, LP10070 conversationnel ...)
- un usager pour le travail en temps réel
- un usager pour le travail de fond
- l'usager PTT chargé des entrées-sorties avec les périphériques lents. Il comporte autant de processus que de périphériques susceptibles de travailler en parallèle, plus quelques processus de gestion.

Il existe une relation entre les états, actif ou bloqué, des processus d'un usager et les états d'un usager :

- un usager est oisif si tous ses processus sont bloqués. L'oisiveté est un état intrinsèque
- un usager est non-oisif dès qu'un de ses processus est actif.

Un usager non oisif passe à son tour par un certain nombre d'états qui sont fonction de l'allocation des ressources. Ces états dont la raison d'être est technologique seront introduits ultérieurement.

1.4 Bibliographie

[1] CII Ordinateur 10070, manuel d'exploitation. Note technique 2444.

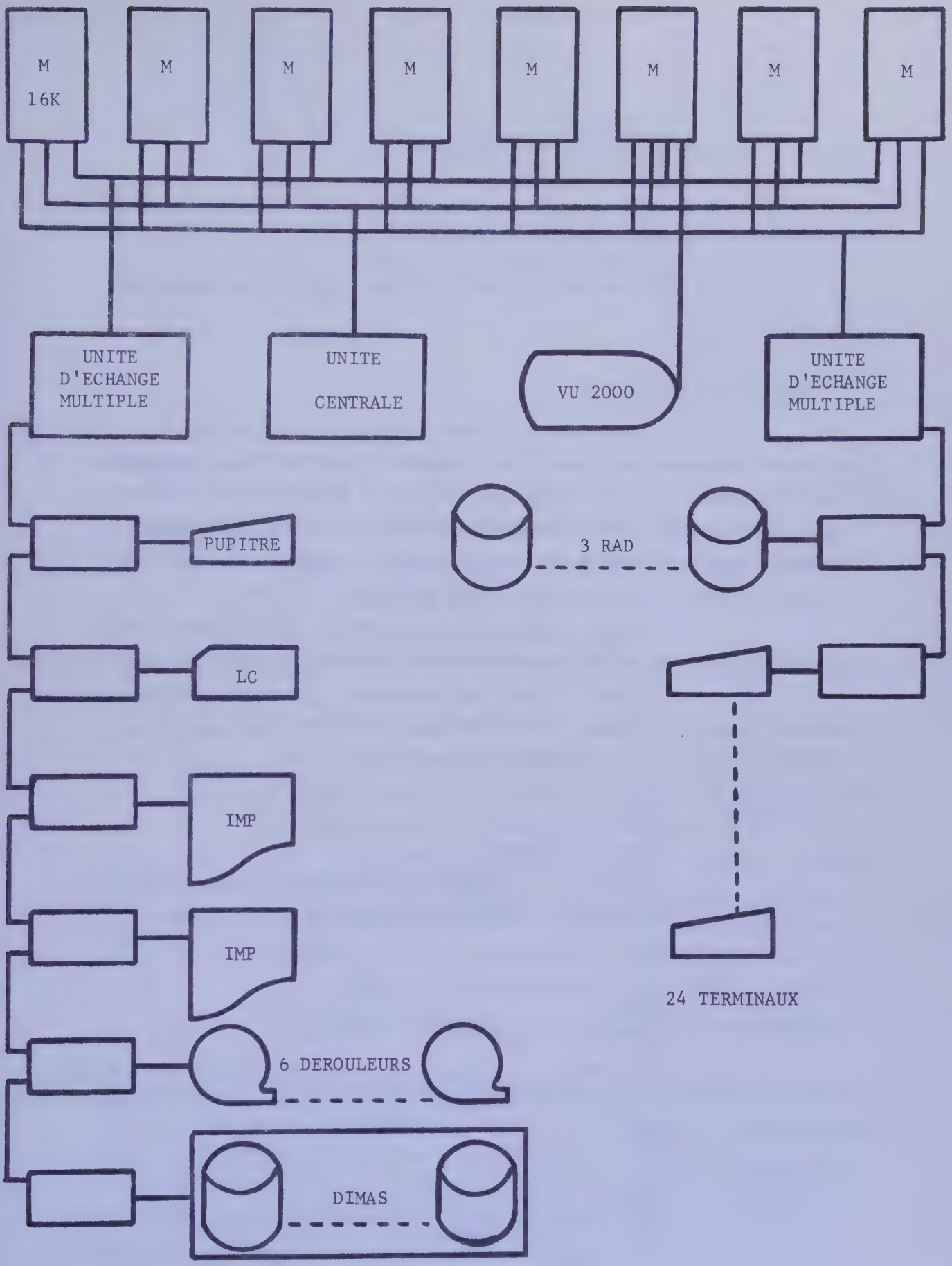


PLANCHE 1 : CONFIGURATION DU SYSTEME 10070 DE L'IRIA

CHAPITRE 2

COOPERATION DES PROCESSUS

Elle repose sur le mécanisme indirect des sémaphores [1, 2].

2.1 Définition des sémaphores

Un sémaphore est constitué par l'association d'une variable entière E et d'une file d'attente F appelée file d'attente du sémaphore. La variable entière E peut prendre des valeurs positives, négatives ou nulles. Les éléments de la file d'attente F, lorsque celle-ci n'est pas vide, sont des processus, ou plus exactement des pointeurs vers des vecteurs d'état de processus.

La modification d'un sémaphore ne peut être faite que par l'intermédiaire d'opérations de base, ou primitives de synchronisation.

Soit sem le nom d'un sémaphore, E (sem) est la variable associée à sem et F (sem) la file d'attente de sem. Les principales primitives jouant sur les sémaphores sont :

```
P(sem) :  begin
          E(sem) := E(sem) -1 ;
          if E(sem) < 0  then  le processus qui exécute cette
          primitive se bloque et rejoint la file F(sem) ;
          end

V(sem) :  begin
          E(sem) := E(sem) + 1 ;
          if E(sem) ⩽ 0  then  un processus de F(sem) est oté de
          cette file et est remis à l'état actif
          end

D(sem) :  begin
          while E(sem) < 0 do
                begin
                E(sem) := E(sem) + 1 ;
```

```
            oter un processus de F(sem) et le rendre actif
            end ;
        E(sem) := 0
        end
```

Ces algorithmes appellent les remarques suivantes.

Remarque 1. Le choix du processus de F(sem) à activer dans V et D n'est pas lié à la primitive elle-même ; il dépend de la politique de priorité définie pour les files d'attente des sémaphores. Selon la manière d'ordonner les éléments de la file, celle-ci peut être par exemple :

- une pile : dernier arrivé, premier servi
- une queue : premier arrivé, premier servi
- une file ordonnée selon une priorité explicite. On a le choix entre 2 politiques :
  - ordonner les éléments dès leur entrée dans la file et toujours servir le premier
  - mettre tout nouvel élément en queue (ou tête) de file et rechercher le plus prioritaire pour le servir.

    Cette seconde politique de parcourir toute la file pour rechercher le plus prioritaire tandis que la première profite de l'ordre existant entre les éléments pour ne parcourir qu'une partie de la file à l'entrée d'un nouvel élément. On préfère d'ordinaire utiliser la première politique qui est plus efficace.

Remarque 2. Le blocage et les activations décrits dans ces primitives sont des opérations intrinsèques. Lorsque la file F(sem) a été modifiée se pose le choix technologique d'attribution de l'unité centrale à un processus actif. Ce choix ne dépend pas de la logique des primitives P, V ou D, mais uniquement de la politique d'allocation de l'unité centrale.

Remarque 3. Les opérations primitives doivent être indivisibles. En conséquence si deux processus tentent d'exécuter simultanément deux de ces primitives (ou la même), les deux opérations ne sont prises en compte et exécutées que l'une après l'autre, dans un ordre qui est

La définition de ces primitives a pour conséquence :

1. Si E(sem) est négative, sa valeur absolue est égale au nombre de processus bloqués dans la file F(sem).

2. Si E(sem) est positif, et si le sémaphore sem sert à contrôler l'accès à une ressource, la valeur de E(sem) donne le nombre de voies d'accès à la ressource qui sont encore libres.

2.2 Exemples d'emploi des sémaphores

2.2.1 Soit une ressource dont on veut contrôler l'accès. C'est par exemple un tableau de données commun à plusieurs processus et modifié par chacun d'eux. Ce problème a été longuement étudié sous le nom de problème de l'exclusion mutuelle ou de la section critique [1] :

- soit deux processus cycliques A et B. Au cours de leur cycle chaque processus traverse une section critique pendant laquelle par exemple se déroule une phase de modification du tableau commun. Le problème consiste à s'assurer que les deux processus ne soient pas en même temps dans leur section critique. La solution ne doit pas faire intervenir les vitesses relatives des deux processus, qui sont quelconques ; elle ne doit pas non plus avoir pour conséquence de synchroniser les processus ; elle ne doit pas enfin permettre le risque que les deux processus se gènent mutuellement et indéfiniment au moment d'entrer dans la section critique.

La solution donnée par T.J. DEKKER et généralisée par E.W. DIJKSTRA au cas de n processus a des inconvénients majeurs :

- elle est complexe et met en jeu de nombreuses variables booléennes
- elle suppose connu et fixe le nombre des processus cycliques
- lorsqu'un processus est bloqué parce qu'un autre est dans sa section critique, il est mis dans une boucle d'attente qui immobilise un processeur. La solution est mal adaptée au cas réel où le nombre des processeurs est inférieur au nombre des processus.

L'emploi des sémaphores est d'une grande simplicité :

- on déclare un sémaphore mutex tel que E(mutex) := 1
- processus A :
```
begin
    L1 : section non critique ;
    P(mutex) ;
    section critique ;
    V(mutex) ;
    goto L1
end
```

- processus B :
```
begin
    L2 : section non critique ;
    P(mutex) ;
    section critique ;
    V(mutex) ;
    goto L2
end
```

- etc.....

Comme le sémaphore mutex est initialisé à un, seul un processus peut s'engager dans sa phase critique. L'autre attend bloqué sur un P(mutex) jusqu'au moment où le processus actif sorte de sa phase critique en exécutant la primitive V(mutex). L'exclusion mutuelle est bien garantie.

Lorsqu'il y a n processus, la valeur minimale que peut atteindre E(mutex) est 1-n, lorsque n-1 processus sont bloqués dans la file F(mutex) pendant que le n^e processus se trouve en section critique.

On peut généraliser le problème de l'exclusion mutuelle pour n processus au cas où k processus au plus peuvent simultanément se trouver dans leur section critique ($k < n$). Il s'agit dans ce cas, de l'accès à une ressource à k points d'accès, comme par exemple un canal d'entrée-sortie avec k sous canaux banalisés. Il suffit d'initialiser le sémaphore de telle manière que E(mutex) = k. On a alors :

$$k \geq E(mutex) \geq k-n$$

2.2.2 Soit deux processus A et B coopérant à une certaine tâche. Au départ, A est actif et B est bloqué. Arrivé à un point de son exécution,

A veut activer B pour que celui-ci exécute sa part de travail. Quand B a terminé celle-ci, il en avertit A, que celui-ci soit actif ou bloqué. Ce schéma est celui où A est un processus de calcul et B un processus d'entrée sortie pouvant se dérouler en parallèle avec le calcul.

Ce schéma a trouvé une solution où intervient le concept d'événement [3, 4]. Il peut également être résolu grâce à l'emploi de sémaphores privés. On appelle sémaphore privé un sémaphore associé à un processus particulier qui est le seul à effectuer sur lui des opérations P. Tout autre processus ne peut exécuter que des opérations V ou D. De plus un sémaphore privé est toujours initialisé à zéro.

La solution est alors comme suit :

- on déclare deux sémaphores privés apriv et bpriv tels que
  E(apriv) := E(bpriv) := 0
- processus A : begin
  A1 ;
  V(bpriv) ;
  A2 ;
  P(apriv) ;
  A3 ;
  end
- processus B : begin
  P(bpriv) ;
  B1 ;
  V(apriv) ;
  end

Le processus B se bloque sur P(bpriv) tandis que A entre dans la phase A1. En fin de celle-ci, il actionne bpriv et réveille B. Les phases A2 et B1 se déroulent en parallèle. Si A2 se termine avant B1, A se bloque sur apriv et il ne sera réveillé qu'en fin de B1. Si au contraire B1 se termine le premier, apriv prend la valeur 1 et A ne sera pas arrêté entre A2 et A3 par l'opération P. On a :

- $-1 \leq bpriv \leq 0$
- $-1 \leq apriv \leq 1$

2.2.3 L'activation du processus C subordonnée à l'émission d'un signal par les processus A et B, se programme :

- on déclare un sémaphore privé sempriv tel que E(sempriv) := 0
- processus A : begin
  A1 ; A2 ;...;
  V(sempriv)
  end

- processus B : begin
  B1 ; B2 ;...;
  V(sempriv)
  end

- processus C : begin
  P(sempriv) ; P(sempriv) ;
  C1 ; C2 ;...;
  end

Alors

- $1 \leqslant$ sempriv $\leqslant 2$

2.2.4 L'activation du processus C subordonnée à l'émission d'un signal par l'un des processus A ou B devient :

- on déclare un sémaphore privé sempriv tel que E(sempriv) := 0
- processus A : begin
  A1 ; A2 ;...;
  V(sempriv)
  end

- processus B : begin
  B1 ; B2 ;...;
  V(sempriv)
  end

- processus C : begin
  P(sempriv) ;
  C1 ; C2 ;...;
  end

On a : -1 < sempriv < 2

2.2.5 D'une façon générale, chaque fois qu'un processus a besoin, pour continuer son évolution, de connaître la valeur de certaines variables d'état qui ont pu être modifiées par d'autres processus, il met en jeu un sémaphore privé.

Ce sémaphore mémorise un événement, celui qui se traduit par la modification significative des variables d'état. La séquence des instructions exécutées par ce processus est alors :

```
P(mutex) ;
if événement arrivé then V(sempriv) ;
V(mutex) ;
P(sempriv)
```

Ainsi lorsque la lecture des variables globales apprend que l'événement n'est pas arrivé, le processus se bloque sur son sémaphore privé, à la sortie de la section critique, laissant aux autres processus le soin d'exécuter une opération V(sempriv) au moment favorable. Ces autres processus effectueront donc la séquence ci-après :

```
P(mutex) ;
modification des variables d'état ;
if événement arrivé then V(sempriv) ;
V(mutex)
```

Bien entendu, les sémaphores sont initialisés de telle façon que

```
E(sempriv) := 0
E(mutex)   := 1
```

2.2.6 Soit deux processus cycliques appelés producteur et consommateur.

A chaque cycle de leur activité, le producteur génère des informations tandis que le consommateur les exploite. Les informations produites ou consommées sont situées dans une zone de tampons accessibles aux deux processus. Toutefois les deux processus ne peuvent travailler simultanément sur le même tampon.

On ne fait aucune prédiction sur les vitesses relatives des deux processus. Le tampon doit servir à amortir la différence possible entre les deux vitesses.

Le producteur remplit des tampons, tant qu'il y en a de vides, et se bloque lorsqu'il n'en reste plus. Le consommateur prélève les tampons pleins qu'il vide jusqu'à ce qu'ils soient tous vides. Dans ce cas il attend. Le problème consiste à programmer ces attentes et à prévoir le réveil du producteur lorsqu'un tampon vient d'être vidé ou du consommateur lorsqu'un tampon vient d'être rempli.

Soit une zone de n tampons. Le problème du producteur et du consommateur se traite alors ainsi :

- on déclare deux sémaphores

  sempro pour bloquer et activer le producteur

  semcons pour bloquer et activer le consommaxeur

  ils sont initialisés de manière que :

  E(sempro) := n

  E(semcons) := o

- processus producteur :

```
begin
    i := 0 ;
E1 : phase de production ;
    P(sempro) ;
    phase de remplissage du tampon i ;
    i := i +1 modulo n ;
    V(semcons) ;
    goto E1
    end
```

- processus consommateur :

```
begin
    j := 0 ;
E2 : P(semcons) ;
    phase de vidage du tampon j
    j := j + 1 modulo n ;
    V(sempro) ;
    phase d'exploitation des informations ;
    goto E2
    end
```

On a : $-1 \leq E(sempro) \leq n$

$-1 \leq E(semcons) \leq n$

$n-2 \leq E(sempro) + E(semcons) \leq n$

On constate que E(sempro) indique à tout moment le nombre de tampons encore vides tandis que E(semcons) indique le total des tampons pleins. On peut avoir au plus deux tampons à moitié vides (ou à moitié pleins).

Le problème du producteur et du consommateur se rencontre typiquement dans le cas des entrées ou sorties d'information en parallèle avec le calcul. Par exemple, le processus producteur est un processus de calcul et il produit des résultats dans un tampon qui est vidé par un processus imprimante.

2.2.7 Soit n processus voyageurs. De temps en temps un certain nombre d'entre eux veulent prendre le métro. Pour cela celui qui manifeste ce désir se rend sur le quai et attend. Quand le métro arrive, tous ceux qui attendent y montent et le métro repart avec les voyageurs. Par hypothèse on sait qu'il y a de la place dans le métro pour tous les n voyageurs à la fois.

Le problème du métro peut se coder comme suit :

- on utilise un sémaphore train initialisé de façon que E(train) := 0
- processus métro : begin
  E1: phase de déplacement du métro et d'arrivée en gare ;
  D (train) ;
  goto E1
  end
- quant à chaque processus voyageur il s'écrit

voyageur : begin
phase de déplacement et de descente dans la station ;
P (train) ;
phase de déplacement en métro
end

Nota : $-n \leq E(train) \leq 0$

Application : Ce problème apparaît dans ESOPE en liaison avec la stratégie de chargement de page à la demande (voir 3e partie ch.1). Lorsqu'une page virtuelle n'est pas en mémoire, la protection correspondante est mise dans l'état "aucun accès". Toute référence à cette page engendre

une trappe appelée trappe pour défaut de page. Cette trappe a pour effet de forcer dans le processus l'exécution d'une opération P de nature technologique (et non intrinsèque) sur un sémaphore S:REQ propre à l'usager. Le moniteur attend que tous les processus d'un usager aient fauté avant de lancer une requête de chargement de l'ensemble des pages absentes. Lorsque cette requête a été satisfaite et que toutes les pages sont en mémoire, tous les processus bloqués derrière S:REQ sont réactivés par une opération D(S:REQ.)

2.2.8 Soit un processus que l'on souhaite activer seulement s'il est endormi et une fois seulement même si plusieurs signaux d'activation sont émis.
Supposons le processus cyclique et contrôlé par un sémaphore privé semprem. Ce problème se code comme suit :

- processus :
```
begin
A : P(semprem) ;
    corps du processus ;
    goto A
    end
```

- signal d'activation : D(semprem)

On vérifie aisément que les conditions sont vérifiées :
- si le processus est bloqué, D(semprem) l'active
- si le processus est actif, D(semprem) n'a aucun effet et ceci quel que soit le nombre de fois que cette opération est faite.

Application : cette situation apparaît dans ESOPE pour le processus premier d'un usager qui doit être réveillé par la touche "appel système" du téléimprimeur. Cette touche, qui n'est jamais bloquée, permet d'arrêter l'exécution d'une commande qui conduirait à une action non souhaitée par l'utilisateur (calcul bouclant indéfiniment, impression d'un texte trop bavard, etc...). En appuyant sur cette touche, on réveille le processus premier qui, étant le plus prioritaire des processus de l'usager, prend le contrôle et arrête tous les autres processus de l'usager.

2.2.9 Ce dernier exemple a pour intérêt de montrer comment on peut traiter un problème complexe de synchronisation à l'aide des primitives P et V. Ce problème porte le nom de problème des spaghettis.[5]

a) Présentation et analyse

Soient 5 personnes notées i (i = 0, 1, .., 4) s'apprêtant à prendre place autour d'une table sur laquelle sont disposés 5 couverts ; des spaghettis sont au menu ! Sachant qu'à chaque couvert est associée une fourchette et une seule, le rituel du repas doit obéir aux exigences suivantes :

a - A tout instant donné chaque personne se trouve dans l'un des deux états suivants : ou bien elle pense ou bien elle mange.

b - Toute personne qui mange a besoin de deux fourchettes (celle de son voisin de gauche et la sienne) de sorte que deux voisins ne peuvent se trouver à table simultanément.

c - Initialement, toutes les personnes pensent.

Ainsi le comportement de chaque convive se réduit à une succession d'intervalles quelconques de réflexion et de ripaille !

Assurer une coopération "harmonieuse" de ces 5 personnes, c'est trouver un mécanisme qui permette d'éviter tout conflit : le cas typique d'un conflit conduisant à "l'étreinte fatale" est celui où tous les convives, ayant décidé de manger, saisissent au même moment leur propre fourchette, interdisant du même coup à leur voisin l'usage de la seconde fourchette ; de plus ce mécanisme doit éviter à toute personne une attente infinie : si un convive qui a décidé de manger trouve la place d'un de ses voisins immédiats occupée (il est alors sûr qu'il lui manquera au moins une fourchette) il continue ses cogitations, mais doit cependant être réveillé aussitôt que ses deux fourchettes deviennent libres.

Les activités (manger ou penser) mises en jeu par chaque personne sont strictement séquentielles et n'ont d'interactions avec les activités des autres personnes qu'au début et à la fin de leur exécution ; en outre, aucune prédiction ne doit être faite sur la vitesse d'exécution de ces actions. Nous pouvons dès lors assimiler le comportement de chaque convive à un processus cyclique qui réalise

d'une façon identique pour tous les activités élémentaires - penser et manger.

Il s'agit donc d'élaborer un mécanisme qui permette à ces 5 processus parallèles de coopérer sans blocage ; nous noterons, tout de suite, que le problème étant parfaitement symétrique, ce mécanisme de synchronisation se retrouvera identique dans chaque processus.

b) Formalisation

Une façon intuitive d'aborder le problème serait de considérer chaque fourchette comme une ressource à un point d'accès et d'appliquer l'exclusion mutuelle de deux voisins pour l'usage de la même fourchette. Notons que les processus étant parallèles, cette manière d'envisager les choses peut conduire à "l'étreinte fatale" déjà citée, lorsque chacun ayant saisi sa propre fourchette, se trouverait en attente de celle de son voisin de gauche ; cette solution est à rejeter.

Considérons donc les deux états dans lesquels chaque convive i peut se trouver :

$c[i] = 0$ lorsqu'il pense,

$c[i] = 2$ lorsqu'il mange.

Dans toute la suite de cet exemple nous utiliserons les notations suivantes :

+ (ou -)	addition (ou soustraction) modulo 5,
$\wedge$	intersection logique.

Avec ces notations, l'hypothèse (b) s'écrit :

$$\nexists i : (c[i] = 2) \wedge (c[i + 1] = 2)$$

- Supposons que le convive i décide de passer de l'état $c[i] = 0$ à l'état $c[i] = 2$.

Ce changement d'état est possible si la condition :

$$(1) \quad (c[i + 1] \neq 2) \wedge (c[i - 1] \neq 2)$$

est réalisée.

La réalisation de cette condition doit entrainer :

- le changement d'état ($c[i] = 2$) immédiat du convive i ;
- la continuation possible de son activité (manger).

Dans le cas contraire, la personne i n'est pas autorisée à passer à l'état $c[i] = 2$.

Cette possibilité (ou impossibilité) d'évolution du processus i peut être matérialisée en associant au processus i un sémaphore privé que nous noterons sempriv[i].

Naturellement le test de la condition (1) et les conséquences qui en résultent, ne doivent être accessibles qu'à une seule personne à la fois, et pour cette raison constituent une section critique protégée par un sémaphore d'exclusion mutuelle noté mutex.

Supposons que le convive i décide de passer de l'état $c[i] = 2$ à l'état $c[i] = 0$.

Ce changement d'état peut entrainer des modifications pour ses deux voisins immédiats s'ils avaient déjà essayé de passer de l'état 0 à l'état 2, et ce afin de leur éviter une attente infinie.

Le réveil du voisin (i + 1) est possible si on est sûr qu'il dispose d'une seconde fourchette autrement dit si $c[i + 2] \neq 2$; il en est de même pour le voisin (i - 1).

A ce niveau, nous sommes obligés d'introduire un état supplémentaire $c[i + 1] = 1$. Cet état qui concrétise la tentative de passage de l'état 0 à l'état 2 du convive (i + 1) ; ainsi $c[i] = 1$ signifie que la personne i a essayé de passer à l'état 2 sans succès (cf. remarque 1).

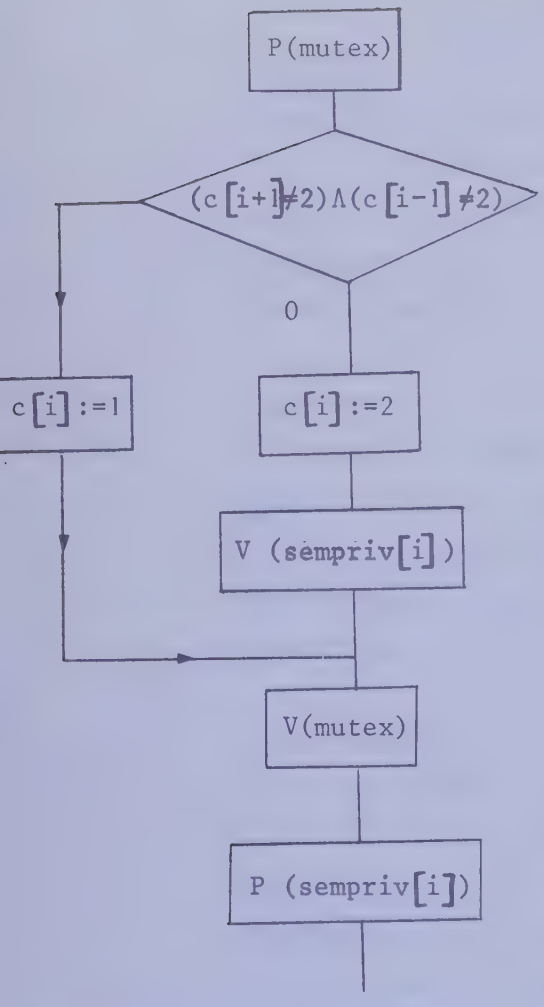
En résumé, le passage du convive i de l'état $c[i] = 2$ à l'état $c[i] = 0$ déclenche les actions suivantes :

- changement immédiat de son propre état ($c[i] = 0$),
- passage du convive (i + 1) à l'état $c[i + 1] = 2$ si la condition $(c[i + 1] = 1) \wedge (c[i + 2] \neq 2)$ est réalisée et continuation possible de l'activité de (i + 1) par une opération du type V sur le sémaphore privé sempriv [i + 1].
- les mêmes opérations sont déclenchées sur le convive (i-1)

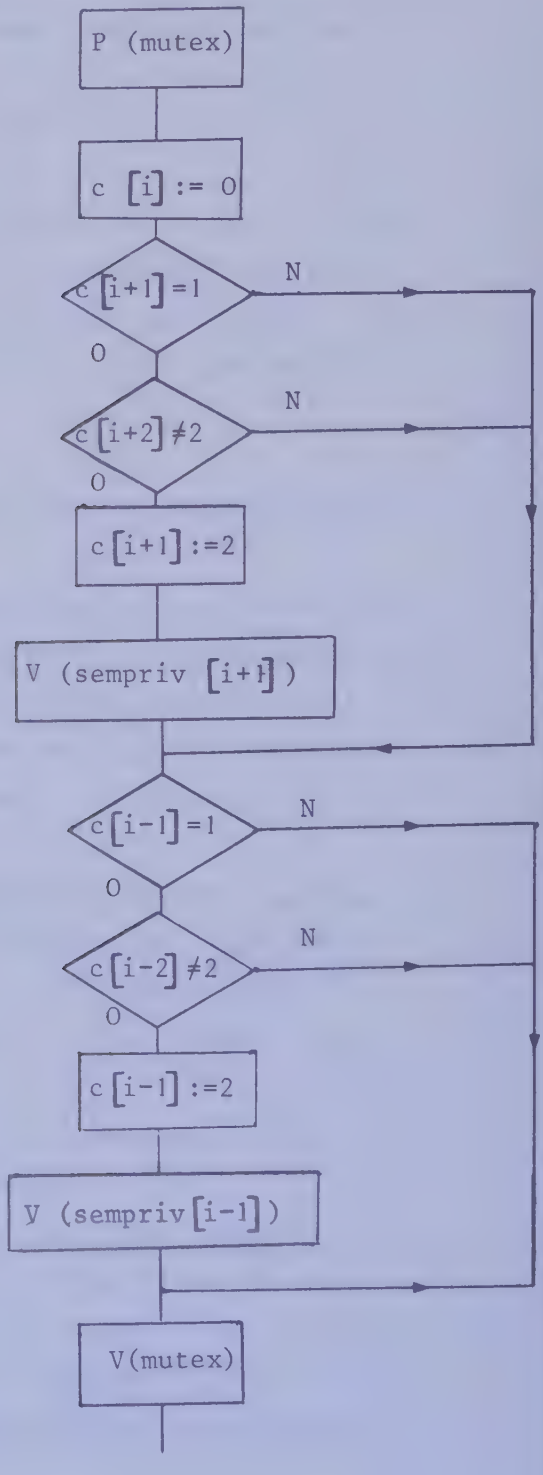
Ces actions constituant une section critique doivent être protégées par le sémaphore d'exclusion mutuelle, mutex.

Remarque 1 : Si on libère le sémaphore privé associé au processus (i + 1) sans tenir compte du fait que ce dernier est ou non bloqué derrière, la synchronisation devient impossible ; Notons que si la lecture de la valeur prise par le sémaphore était possible, l'introduction du nouvel état c[i]= 1 serait superflue.

Nous donnons, ci-dessous, une représentation linéaire du mécanisme de synchronisation pour le processus i ; ce mécanisme se retrouve identique dans les autres processus.



a. Passage de l'état c[i] = 0 à l'état c[i] = 2



b. Passage de l'état c[i] = 2 à l'état c[i] = 0

c) Algorithme

En pseudo-algol, l'algorithme de synchronisation s'écrit :

```
begin array c[0 : 4]; integer j ;
      sémaphore array sempriv[0 : 4]; comment sémaphores privés ;
      sémaphore mutex ; comment sémaphore d'exclusion mutuelle ;
      procédure test (k) ; value k ; integer k ;
                if c[k] = 1 ∧ c[k + 1] ≠ 2 ∧ c[k - 1] ≠ 2
                then begin c[k] : = 2 ; V (sempriv[k]) end ;
      for j : = 0 step 1 until 4 do
                c[j] : = sempriv[j] : = 0 ;
      mutex : = 1 ;

      parbegin
          :
personne i : Li : begin
                  pense ;
                  P (mutex) ; c[i] : = 1 ; test (i) ; V (mutex) ;
                  P (sempriv[i]) ; mange ;
                  P (mutex) ; c[i] : = 0 ; test (i - 1) ; test (i+1) ;
                  V (mutex) ;
                  go to Li
                  end
          :
      parend
end
```

Nota : On utilise ici la notation parbegin ... parend pour indiquer que les blocs qui sont compris entre ces deux délimiteurs correspondent au code de processus parallèles.

2.2.10 Essayons d'exprimer l'opération D(sem) en fonction de P et de V.

Soit un certain nombre de processus A1, A2,... An qui se bloquent sur un P(sem) et soit un processus libérateur L qui libère tous ceux qui sont bloqués à l'aide d'un D(sem) et ceux là seulement. E(sem) est initialisé à 0.

Supposons que P(sem) et D(sem) soient remplacées par les opérations suivantes :

- on déclare un sémaphore mutex, initialisé à 1, une variable auxiliaire t, initialement à 0 et un sémaphore sem initialisé à 0.
- dans chaque Ai, on remplace P(sem) par

```
P(mutex) ;
t := t + 1 ;
V(mutex) ;
P(sem) ;
```

- dans L, on remplace D(sem) par

```
P(mutex) ;
for k := 1 step 1 until t do V(sem) ;
t := o ;
V(mutex) ;
```

La variable t est utilisée pour compter le nombre des processus bloqués ou en passe de l'être. Cette programmation simple est fausse car elle permet la séquence suivante :

```
Ai fait P(mutex) ;t := t + 1 ; V(mutex) ; comment : t = to, E(sem)=to-1 ;
 L fait l'équivalent de D(sem) ; comment : t = o, E(sem) = 1 ;
Aj fait P(mutex) ;t := t + 1 ; V(mutex) ; P(sem); etc...; comment t=1,
                                                    E(sem) = o ;
Ai fait P(sem) ; comment Ai se bloque alors que Aj est passé à sa place ! ;
```

Pour que le programme soit exact, il faut associer une variable ti et un sémaphore privé sem i à chaque processus Ai. On constate la simplification apportée par l'emploi du D.

2.2.11 On trouvera d'autres exemples de problèmes dans [6].

2.3. Réalisation des mécanismes

Plusieurs problèmes doivent être résolus dans le cadre d'une implantation des sémaphores en machine.

2.3.1 Tout d'abord l'indivisibilité des primitives doit être assuré. Dans le cas d'ESOPE où nous avons voulu simuler une machine à sémaphore, toute interruption génère une primitive V ou une primitive D. En conséquence le code de toute primitive doit être protégé contre toute interruption.

Notons que dans le cas d'un multiprocesseur ceci ne suffirait pas : il faudrait interdire encore à deux ou plusieurs processeurs d'exécuter ensemble une primitive. Ceci ne peut se faire que grâce à l'emploi d'une instruction ininterruptible "test and set lock" (TSL) qui opère comme suit :

- regarde si la valeur de la cellule donnée en adresse est positive ou nulle,
- si oui la rend négative et saute une instruction,
- si non passe en séquence.

L'exécution d'une primitive est précédée du code :

```
TSL    mem
B      *-1      (branchement à l'instruction précédente)
```

et se termine par :

```
LI,R   1        (chargement du registre R avec la valeur 1)
STW,R  mem      (écriture de R dans mem)
```

On aura reconnu des opérations analogues à P et à V, jouant au niveau des primitives et protégeant leur code, considéré comme une section critique. Toutefois à la différence de P, lorsqu'il y a déjà un processeur en section critique (mem négatif), les autres font une boucle d'attente.

2.3.2 Les noms de sémaphore doivent-ils être connus de tous les processus (donc communs à chacun d'eux) ou doit-on restreindre leur portée ? La réponse dépend du système dans lequel sont utilisés les sémaphores. WIRTH ([7]) propose une implémentation dans le langage PL 360 sur l'IBM 360, où les processus se partagent un seul espace

virtuel et où les sémaphores sont repérés par leur adresse dans cet espace. Comme le PL 360 possède la structure de bloc (mais sans allocation dynamique), deux sémaphores dans des blocs extérieurs l'un à l'autre peuvent avoir le même nom ; ils ont néanmoins des adresses différentes.

Dans ESOPE, intervient à ce propos la notion d'usager. Nous l'avons définie comme une famille de processus. Nous verrons plus loin que nous associons à chaque usager une mémoire virtuelle et que l'une des propriétés de la mémoire virtuelle est d'être isolée des autres mémoires virtuelles. Deux programmes dans deux mémoires virtuelles différentes sont donc protégés, l'un contre les erreurs de l'autre. Nous gardons cette propriété pour les sémaphores :

Règle : les sémaphores d'un usager ne sont pas accessibles à un autre usager.

Cette règle s'appliquerait bien si les usagers étaient tous technologiquement indépendants. Ce n'est pas le cas, aussi y-a-t'il quelques exceptions à la règle. Regardons en détail les divers usagers et les coopérations qui existent entre eux.

Le moniteur possède plusieurs processus qui coopèrent entre eux. Dans certains cas, il doivent coopérer avec les processus des usagers,

- à l'entrée d'une session à un téléimprimeur, le moniteur créé un usager, puis lance son processus premier, chargé d'interpréter le langage de commande. Ce processus est activé par une primitive V sur son sémaphore privé. Tous les sémaphores privés des processus premiers sont donc connus du moniteur.
- en fin de session, le processus premier de l'usager qui termine alerte à son tour le moniteur pour que l'usager soit détruit. Il le fait par une primitive V sur un sémaphore privé du moniteur. Un sémaphore du moniteur est donc connu de tous les processus premiers.
- à chaque défaut de page, le processus en cause est bloqué derrière un sémaphore technologique. Le programme de trappe pour défaut de page se termine par une opération P (S:REQ). Le sémaphore S:REQ est unique pour tous les processus de l'usager. Lorsque tous les autres processus d'un usager se sont bloqués intrinsèquement ou technologi-

quement, un processus spécial de l'usager est automatiquement appelé par le mécanisme d'allocation de l'unité centrale. Ce processus appelé processus final alerte par une primitive V le processus du moniteur chargé de lancer la requête de transfert des pages manquantes. un sémaphore du moniteur est donc connu de tous les processus finaux. Réciproquement, lorsque la requête est satisfaite, un processus du moniteur effectue une primitive D(S:REQ), ce qui a pour effet de réactiver tous les processus précédemment bloqués pour défaut de page. Le moniteur connait donc tous les sémaphores S:REQ des usagers.

Les usagers coopèrent avec le moniteur comme on vient de le voir, et coopèrent entre eux chaque fois qu'ils accèdent en écriture à un fichier partagé :

- c'est le cas des sorties sur imprimante. Les usagers mettent des messages de sortie dans un fichier boite à lettre. Comme ce fichier est commun à tous il doit être protégé par un sémaphore d'exclusion mutuelle, connu de tous les usagers. Par ailleurs, chaque fois qu'un message est mis, l'usager PTT est alerté. Le sémaphore correspondant de PTT est donc connu de tous les autres usagers.

Nouvelle formulation de la règle : les sémaphores intrinsèques d'un usager ne sont pas accessibles directement à un autre usager.

2.3.3. Peut-on ranger les variables et les files associées à un sémaphore d'un usager dans la mémoire virtuelle de cet usager ? c'est possible si les processus d'un usager sont bloqués derrière des sémaphores du même usager. Ce n'est pas le cas pour le sémaphore d'exclusion mutuelle du fichier boite à lettre. Par ailleurs, les opérations V et D se font d'un usager à un autre donc d'une mémoire virtuelle à une autre. Il faudrait donc que toutes les variables et files associées pouvant être modifiées par un V ou un D soient résidentes en mémoire.

Il nous a de ce fait paru préférable, parce que plus simple, de ranger toutes les tables de sémaphore et de processus en mémoire physique et de les adresser sans mémoire topographique.

Nous pouvons maintenant présenter la solution employée pour ESOPE.

On distingue deux classes de sémaphores, ceux qui sont utilisés pour résoudre les problèmes technologiques d'utilisation des ressources du système et ceux qui sont utilisés pour les problèmes intrinsèques aux usagers.

Dans la première classe se trouvent tous les sémaphores du moniteur, tous les sémaphores de communication entre le moniteur et les processus premiers (donc tous les sémaphores privés des processus premiers) et tous les sémaphores du défaut de page.

Dans la seconde apparaissent tous les sémaphores créés par le processus premier, par les traducteurs ou par les programmes d'application.

Les sémaphores de la première classe sont utilisés dans des parties du système : moniteur, processus premier, traitement de trappes, qui ont été codées comme un tout. On peut donc associer à chacun de ces sémaphores un indice lors de la génération du système et se servir de cet indice pour les repérer.

Les sémaphores de la seconde classe apparaissent chez les usagers, à des instants indépendants. On ne peut supposer une concertation entre les usagers pour se répartir les indices de sémaphore à utiliser. Nous avons donc choisi de laisser les usagers donner des noms quelconques à leurs sémaphores. Deux usagers peuvent utiliser le même nom, nous aurons néanmoins deux sémaphores différents. Une conversion en indices différents sera faite par le système.

De ces considérations découle la structure employée : tous les sémaphores sont regroupés dans une table unique, dite table des sémaphores. Les sémaphores sont répérés par leur indice dans cette table. Tous les sémaphores de la seconde classe et d'un même usager sont chaînés entre eux. Un champ comprend le nom donné par l'usager à chacun d'eux.

Deux types de primitives ont été réalisés :

- celles qui actionnent des indices de sémaphores. Elles sont rapides. Elles sont réservées aux processus du système (moniteur, plus processus premiers),
- celles qui actionnent des noms de sémaphores. Elles doivent faire la correspondance entre le nom et l'indice. Elles sont réservées aux usagers.

Pour assurer l'indivisibilité des primitives, nous avons utilisé l'instruction d'appel au superviseur, CAL 3, de la 10070. Ce CAL 3 permet de changer le mode de fonctionnement du calculateur et de passer de :

- mode esclave, avec topographie et interruptions autorisées,

à un nouveau fonctionnement, pour la durée de la primitive, en :

- mode maître, sans topographie et interruptions masquées.

Comme la table des sémaphores est commune à tous les processus, le phénomène du blocage (voir chapitre 4.2.5) est possible. On verra qu'on peut s'en protéger avec l'algorithme du banquier.

2.4. Liste des principales primitives

primitives actionnant des indices

P, V, D

E.PS, E.VS, E.DS, comme P,V,D, utilisées dans les sous programmes d'interruption.

primitives actionnant des noms

demander : même effet que P

libérer : même effet que V

délivrer : même effet que D.

primitives modifiant le nombre des sémaphores

déclarer : crée un sémaphore chez l'usager en y associant un nom et une valeur initiale positive ou nulle.

effacer : supprime le sémaphore désigné de la liste des sémaphores de l'usager.

primitives modifiant le nombre des processus

Tous les processus ne sont pas définis à la génération du système. On utilise alors :

créer : crèe un processus chez l'usager en lui associant un nom, une priorité et l'adresse de sa première instruction.

détruire : ôte le processus désigné de la liste des processus de l'usager.

disparaître : le processus actif qui exécute cette primitive se fait hara kiri et demande à être oté de la liste des processus de son usager.

primitives de consultation

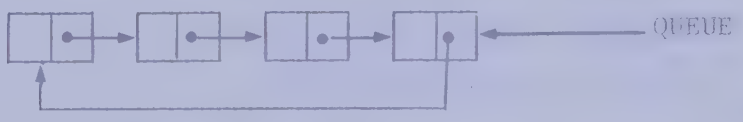
regarder : fournit la valeur du sémaphore désigné par son nom

connaître : permet de lire ou de modifier le mot d'état programme et les registres du processus désigné.

2.5. Structure de données associée

Les informations relatives aux usagers, aux processus et aux sémaphores sont rangées dans des tableaux. L'usager, le processus ou le sémaphore sont particularisés par la valeur de l'indice qui leur correspond.

Certaines informations sont gérées sous forme de file. Les pointeurs utilisés dans ces files contiennent donc des indices dans les tableaux respectifs. Les files utilisées sont toutes de même nature : elles sont circulaires et simplement chaînées. La file est repérée par un pointeur de queue de file



Une file vide est représentée par la valeur NIL dans le pointeur de queue de file.

Cinq files sont utilisées :

- la file associée à un sémaphore
  - pointeur de queue PTPROFIL, élément de la table des sémaphores
  - champ successeur SUCFILSEM, élément de la table des processus
- la file des processus d'un usager
  - pointeur de queue PTPROUS, élément de la table des usagers
  - champ successeur SUCUSRPRO, élément de la table des processus
- la file des indices de processus disponibles
  - pointeur de queue PROVID, variable globale
  - champ successeur SUCUSRPRO, élément de la table des processus
- la file des indices de sémaphore disponibles
  - pointeur de queue SEMVID, variable globale
  - champ successeur, SUCUSRSEM, élément de la table des sémaphores

Dans les tables des usagers, des processus et des sémaphores, on trouve les principaux éléments suivants où ⟹ désigne une queue de file et où → désigne un pointeur vers le successeur :

table des usagers

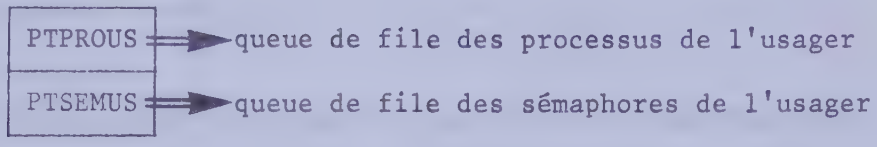
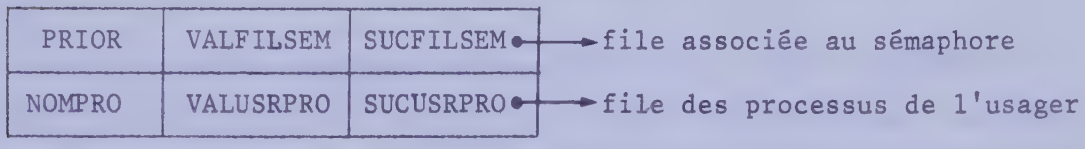
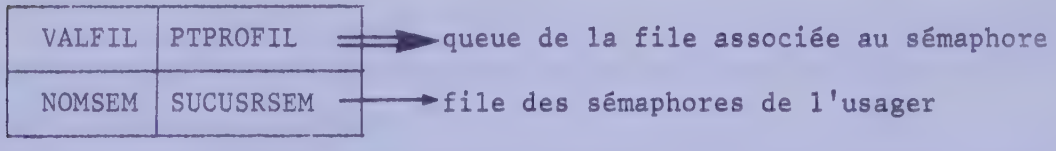


table des processus



VALFILSEM indique dans quelle file de sémaphore se trouve le processus
VALUSRPRO indique à quel usager appartient le processus
PRIOR est la priorité du processus
NOMPRO est le nom que l'usager a donné au processus

table des sémaphores



VALFIL est l'entier associé au sémaphore. C'est ce que nous avons appelé E(sem) au paragraphe 2.1.
NOMSEM est le nom que l'usager a donné au sémaphore.

2.6. Bibliographie

[1] DIJKSTRA. Cooperating sequential processes. EWD 123. T.U. Eindhoven (1965), publié par GENUYS dans Programming languages. Academic Press (1968)

[2] DIJKSTRA. The structure of the THE multiprogramming system CACM 11,5 (mai 1968)

[3] SALTZER. Traffic control in a multiplexed computer system MAC TR30 (juillet 1966)

[4] LAMPSON. A scheduling philosophy for multiprocessing systems CACM 11,5 (mai 1968)

[5] DIJKSTRA. Hierarchical ordering of sequential processes. Acta Informatica (1971)

[6] KAISER. Quelques problèmes de parallélisme et leurs solutions par sémaphore. Société mathématique de Bretagne, Actes des séminaires de Rennes (1972)

[7] WIRTH. On multiprogramming, machine coding and computer organization CACM 12,9 (sept. 1969).

CHAPITRE 3

ACCES A L'INFORMATION - MEMOIRE VIRTUELLE

3.1 Introduction

Deux problèmes contradictoires sont à résoudre dans un système comme ESOPE. D'une part chaque usager indépendant doit être protégé contre des actions intempestives des autres usagers, d'autre part, il est nécessaire de pouvoir partager des programmes et des données.

L'indépendance est atteinte au moyen d'une mémoire virtuelle obtenue par un dispositif de pagination à un niveau appelé mémoire topographique. On associe à chaque usager du système une telle mémoire virtuelle.

Le partage d'information se fait également en utilisant le dispositif de pagination et passe par l'intermédiaire d'un système de fichiers.

3.2 Mémoire virtuelle et pagination sur la 10070

Rappelons que l'information n'a d'existence qu'associée à un support physique. Nous devons donc prendre en considération le couple (information, support).

Le seul support en communication directe avec l'unité centrale est la mémoire principale. Dans celle-ci, les supports élémentaires ou cellules sont repérés par des numéros : ce sont les adresses physiques.

Pour utiliser une information, il faut un moyen de le distinguer des autres informations accessibles au programme. On établit pour cela un classement entre elles et on associe chaque information à un numéro ou adresse virtuelle. Lorsque l'unité centrale effectue un calcul d'adresse en utilisant la partie adresse de l'instruction, l'index et l'indirection, le résultat obtenu est l'adresse virtuelle de l'information. L'ensemble des adresses virtuelles qui peuvent être créées par ce calcul d'adresse porte le nom de mémoire virtuelle.

Le couple (information, support) devient alors le couple (adresse virtuelle, adresse physique).

L'utilisateur d'un calculateur ne s'intéresse qu'à l'information et non au support ; il ne connait que sa mémoire virtuelle, celle de son programme. Un mécanisme spécial fonctionnant à son insu relie les deux éléments du couple (information, support), fait passer de sa mémoire virtuelle à la mémoire physique et achève ainsi l'accès à l'information.

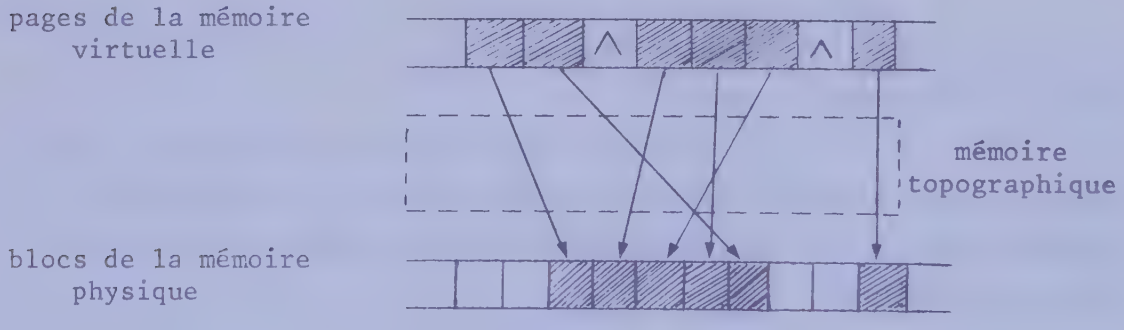
Nous allons décrire ce mécanisme pour le CII-10070. Il porte le nom de pagination et utilise une mémoire spéciale à circuits intégrés, dite mémoire topographique.

La mémoire virtuelle est découpée en pages de 512 mots. Pour cela l'adresse virtuelle de 17 bits est divisée en deux parties :

- les 8 bits de forts poids donnent le numéro de page virtuelle ($2^8 = 256$)
- les 9 bits de faibles poids donnent le déplacement dans la page ($2^9 = 512$)

La mémoire physique est découpée en pages physiques, ou blocs, de même longueur. Les numéros de pages physiques sont aussi codés par 8 bits.

Le mécanisme de pagination fait correspondre à un numéro de page virtuelle un numéro de bloc ou rien (représenté par $\wedge$ sur la figure).



Avant tout accès à l'information le numéro de page virtuelle est remplacé par le numéro de bloc associé et le nombre obtenu sert à adresser la mémoire physique. Comme cette opération est fréquente (environ deux fois par instruction), elle doit être rapide. Le mécanisme de pagination est constitué par une mémoire à

registres adressée par le numéro de page virtuelle. Le contenu de chaque registre indique le bloc associé. Il y a de ce fait autant de registres que de pages virtuelles (256).

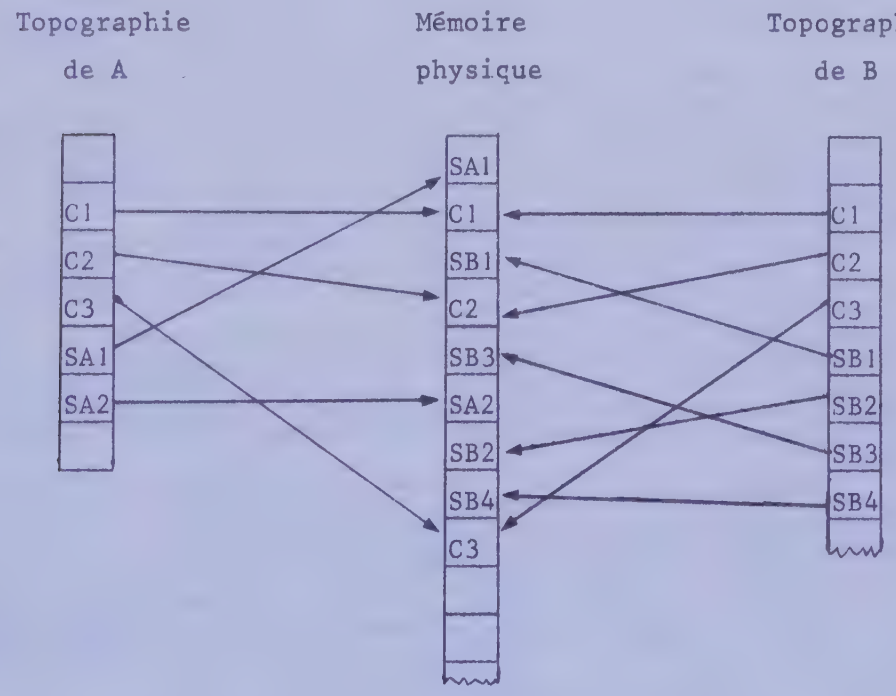
Ce mécanisme a été introduit pour faciliter l'allocation de la mémoire physique. Nous indiquerons son emploi dans les chapitres correspondant.

Nous allons montrer ici dans quelle mesure il permet de résoudre la protection et le partage de l'information.

Le problème de la protection entre deux usagers est résolu en associant une mémoire virtuelle à chaque usager. Il n'y a alors aucune communication possible entre deux blocs associés à deux mémoires virtuelles différentes.

La pagination permet aussi le partage de programme et de données quand un même bloc se trouve associé à deux pages de deux mémoires virtuelles différentes.

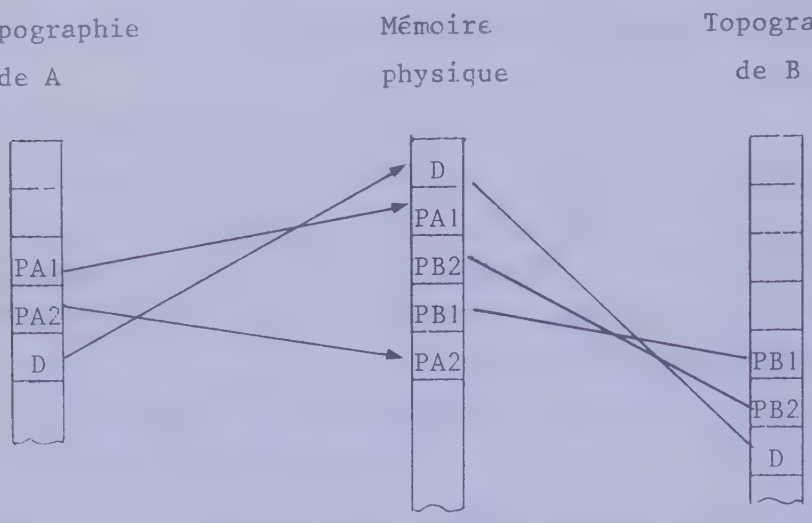
La figure suivante montre comment deux usagers A et B peuvent se partager un compilateur réentrant C :



Les pages du compilateur sont C1, C2 et C3. Ces pages doivent figurer dans chaque mémoire virtuelle à la même adresse. Les programmes sources SA et SB doivent être placés à partir des mêmes adresses virtuelles. En effet, quand le compilateur est chargé dans le système, ses références en mémoire sont associées à des adresses fixes dans la mémoire virtuelle. Chaque fois que le compilateur est utilisé, il fait référence

aux mêmes adresses virtuelles, quel que soit l'usager en cause. Le chargement du compilateur dans la mémoire virtuelle est statique.

La figure ci-après montre le partage de données D par deux procédures PA et PB.



La page partagée D peut être à des emplacements différents dans chaque mémoire virtuelle si les données ne contiennent pas d'adresses. Par contre, si des mots de D sont utilisés pour faire de l'adressage indirect dans PA et PB, ceux-ci doivent être situés aux mêmes adresses en mémoire virtuelle. De même si des mots de D sont des adresses indirectes vers d'autres mots de D, c'est D cette fois qui doit avoir même adresse dans les deux mémoires virtuelles, PA et PB pouvant être à des adresses virtuelles différentes.

Le partage d'information dans une mémoire paginée entraine les contraintes suivantes :

- on doit réserver dans chaque mémoire virtuelle les numéros des pages partagées. La place disponible dans la mémoire virtuelle diminue d'autant,
- on ne peut utiliser simultanément plusieurs procédures partagées qui occupent les mêmes adresses en mémoire virtuelle. Il faut soit permettre une sorte de structure d'"overlay" en mémoire virtuelle, soit effectuer une gestion des mémoires virtuelles de façon à réserver des places différentes pour chaque procédure.

Notons en passant que la structure d'overlay, de même que la présence de procédures réentrantes partagées, ne sont guère compatibles avec une méthode de "swapping" analogue à celle de CTSS ([1]). Nous reviendrons plus loin sur ce problème.

La mémoire virtuelle obtenue par pagination n'offre par ailleurs aucun secours aux problèmes de gestion des structures de données dont la taille varie en cours d'exécution : listes, tables de symboles, piles ou arborescences.

La pagination ne procure qu'un mécanisme d'allocation du support physique. Elle n'apporte aucun dispositif de translation ni d'allocation de place à l'intérieur d'une mémoire virtuelle. On retrouve tous les problèmes classiques d'allocation de mémoire au niveau de la mémoire virtuelle.

L'allocation dynamique de la mémoire virtuelle n'est possible qu'avec une mémoire virtuelle segmentée telle qu'on la trouve dans MULTICS. Nous n'avions pas à notre disposition un tel mécanisme de segmentation, aussi renvoyons-nous pour l'étude de la segmentation à [2] et [3].

3.3 Mémoire virtuelle et usager

Nous avons choisi d'associer une mémoire virtuelle à un usager et non pas à un processus comme il est courant dans la plupart des systèmes de temps partagé.

Cette solution présente l'avantage de diminuer le nombre de topographies à conserver en mémoire. On appelle topographie la table de correspondance page virtuelle - bloc physique. Il y a donc autant de topographie que d'usager et non plus autant que de processus. Par ailleurs, chaque fois qu'on change l'attribution de l'unité centrale, il faut charger la mémoire topographique avec la nouvelle topographie concernée. Si comme on le verra plus loin, l'allocation d'unité centrale est également faite à l'usager, les chargements de la mémoire topographique, qui sont d'une durée non négligeable, sont moins fréquents.

Un autre avantage réside dans la communication d'information entre processus d'un même usager. Comme ils partagent la même mémoire virtuelle, cette communication est immédiate. On n'a pas à trafiquer les topographies.

Par contre, comme la 10070 n'a pas de dispositif de translation d'adresse virtuelle et que les registres d'index sont nécessaires pour l'accès au demi-mot et à l'octet, la réalisation de processus réentrants à l'intérieur d'un même usager est compliquée. Cette gêne est minime. Elle se manifeste surtout au niveau des

procédures du système qui sont disponibles en virtuel. On est parfois amené pendant leur utilisation à nier le parallélisme entre des processus d'un même usager et à protéger ces procédures du système par des sémaphores d'exclusion mutuelle.

Un autre inconvénient dû à la 10070 tient au fait que les processus ne peuvent pas être protégés les uns par rapport aux autres. Le mécanisme de protection est en effet lié à la mémoire virtuelle et non aux processus. Pour éviter cet inconvénient, il faudrait un dispositif de protection par clé et verrou qui fonctionne pour l'accès aux pages virtuelles.

3.4 Le couplage

3.4.1 Définition

Dans certains systèmes, les pages virtuelles qui ne sont pas en mémoire principale sont rangées en mémoire secondaire dans un espace associé à chaque mémoire virtuelle. C'est la mémoire de "swapping" ou mémoire d'instance. Lorsqu'un accès est fait à une page qui n'est pas en mémoire principale, elle est transférée depuis la mémoire d'instance. A la fin de son emploi, elle regagne la mémoire d'instance.

Cette solution a pour inconvénient d'immobiliser une grande place en mémoire secondaire puisque la mémoire d'instance de chaque usager doit pouvoir contenir jusqu'à 128 K mots. Par ailleurs, si la mémoire secondaire contient aussi des fichiers, on est obligé d'avoir deux mécanismes de transfert entre mémoire principale et mémoire secondaire

- un mécanisme rapide pour la mémoire d'instance installée à des adresses fixes
- un mécanisme souple pour les fichiers de taille et d'emplacement variables.

Enfin, cette solution pose des problèmes dans le cas des opérations d'entrée-sortie.

En effet, du point de vue du programmeur, une entrée-sortie est un transfert d'information entre un fichier et sa mémoire virtuelle ; pratiquement, ce transfert ne peut avoir lieu que si les pages correspondantes de mémoire virtuelle sont chargées en mémoire physique. Que doit-on alors faire si l'algorithme d'allocation impose la libération de pages pour lesquelles un transfert est en cours :

- attendre la fin des transferts, ce qui peut être long,
- arrêter les transferts pour les reprendre lors d'une phase ultérieure, ce qui est compliqué et laisse les fichiers dans une situation mal définie,
- utiliser pour les entrées-sorties des tampons résidents en mémoire centrale, mais le nombre de ces tampons croît avec le nombre des usagers et devient rapidement prohibitif ?

C'est pourquoi, nous avons introduit un mécanisme où les pages non associées à des blocs de mémoire principale sont conservées dans le fichier d'où elles sont extraites. Il en résulte plusieurs conséquences :

- les fichiers doivent être découpés en pages de même taille que les pages de la mémoire virtuelle. L'article d'un fichier a la taille d'une page,
- à toute page virtuelle, on doit associer un nom de fichier et un numéro d'article dans lequel la page est rangée ou le sera lorsqu'elle quittera la mémoire principale.

Nous avons aussi admis que le support d'un article de fichier peut être aussi bien une page de mémoire principale qu'une page de mémoire secondaire (ici, les disques RAD). En conséquence :

- le mécanisme de défaut de page utilisé d'habitude pour transférer le contenu d'une page virtuelle de mémoire secondaire à mémoire principale sert aussi à transférer de mémoire secondaire à mémoire principale un article de fichier,
- la mémoire virtuelle d'un usager devient un sous-ensemble de l'espace des fichiers (ou espace virtuel).

Avant ces décisions, l'accès à l'information utilisait le couple (page virtuelle, bloc physique) et dans le cas où il n'y avait pas de page physique associée, on faisait appel au couple (page virtuelle, page de fichier d'instance). Désormais, on considère dans tous les cas successivement les couples suivants :

(page virtuelle, article de fichier)
(article de fichier, support)

Le passage de la mémoire virtuelle aux fichiers se fait au niveau logique, sans mettre en jeu de support physique, par une opération que nous avons appelée couplage ([4], [5]). Le couplage associe un article de fichier - c'est à dire de l'information - à une page virtuelle - c'est à dire un support d'adressage. Comme la mémoire virtuelle contient 256 pages, on sélectionne ainsi un sous ensemble de 256 articles de

fichiers au maximum. Lorsque l'unité centrale veut accéder à l'un d'eux grâce à l'adresse virtuelle qui lui est couplée, elle le fait via le dispositif de mémoire topographique. Si l'article n'est pas en mémoire principale, le mécanisme de défaut de page entre en action et provoque le changement de support.

Cette méthode a été employée pour la première fois dans MULTICS ([6]) dans une optique quelque peu différente puisqu'elle y met en correspondance le segment et le fichier (et non pas la page et l'article) et que les numéros de segment sont alloués dynamiquement (tandis qu'ici les numéros de page virtuelle sont désignés à la compilation).

3.4.2 Structure de données

La localisation d'un article de fichier est obtenue par l'utilisation des tables suivantes :

Table des fichiers opérationnels (TFO)

Tout fichier dont les articles sont situés sur disque RAD ou en mémoire principale possède un numéro interne F, attribué à l'arrivée sur RAD, et qui sert d'index d'entrée dans la TFO. Cette entrée contient en particulier un pointeur PTTIF vers une table décrivant l'implantation du fichier en mémoire, ou TIF.

Table d'implantation du fichier (TIF)

Il existe une table pour chaque fichier ayant une entrée dans la TFO ; le numéro d'article sert d'entrée dans cette table. A chaque entrée on trouve :

- l'emplacement original de l'article sur le disque RAD (locdisq)
- le numéro de la page physique dans laquelle se trouve éventuellement chargé l'article (ψ =0 indique que l'article n'est présent que sur disque RAD).

Table des pages virtuelles (TPV)

A chaque mémoire virtuelle, donc à chaque usager, est associé une table, dite table des pages virtuelles, définissant sa composition. Le numéro de page virtuelle sert d'index d'entrée.
On y trouve.

- le numéro de fichier F et le numéro de l'article couplé à la page virtuelle (F = 0 indique l'absence de couplage),
- une clé PF fixant la protection de la page demandée par l'utilisateur.

Le système hardware de protection des pages virtuelles de la 10070 permet de contrôler l'accès à une page suivant 4 modes d'utilisation :

0 - tout accès permis (écriture, exécution, lecture)

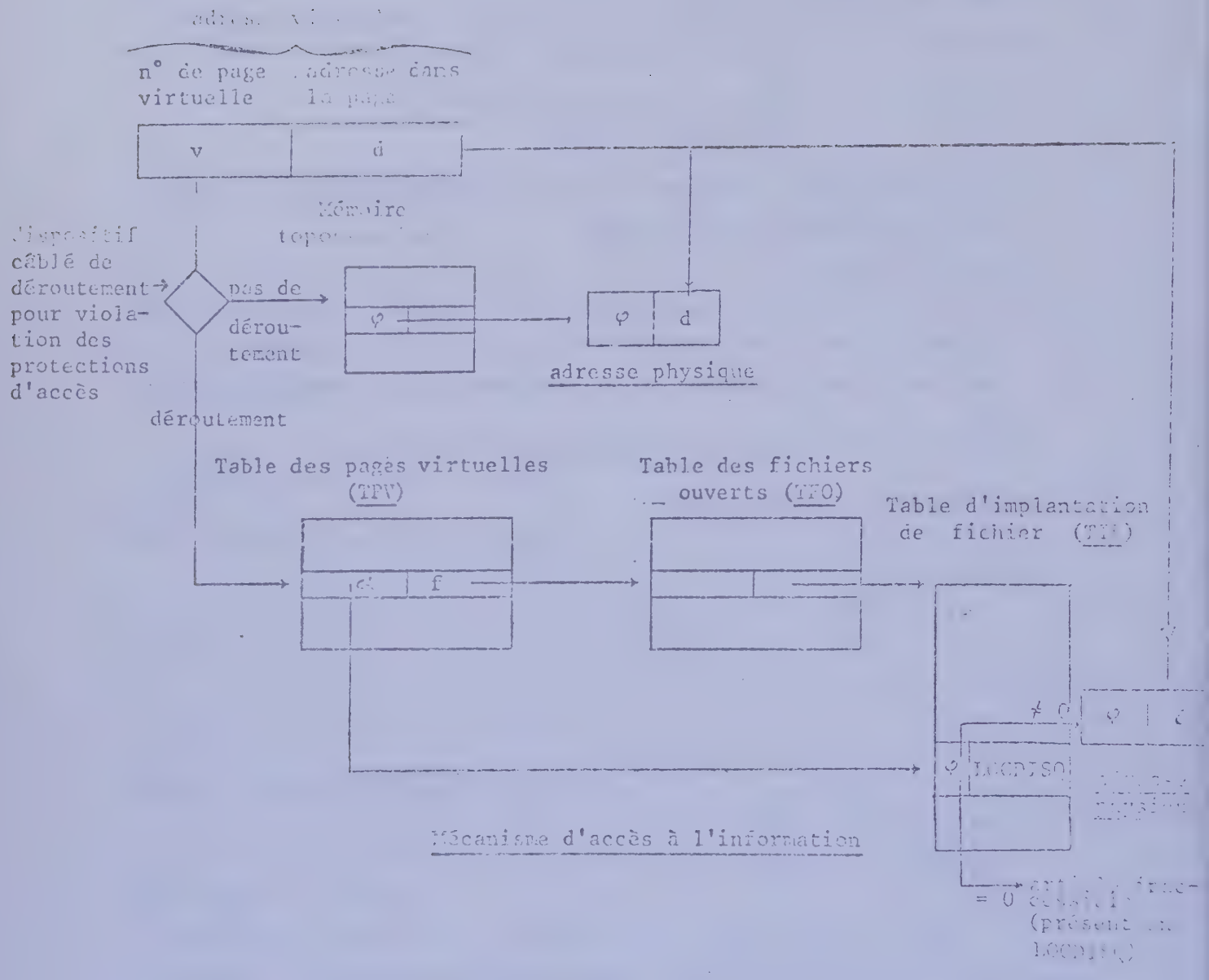
1 - seule l'écriture est interdite

2 - seule la lecture est permise

3 - tout accès interdit

- une clé PE fixant la protection affectée dynamiquement à la page :
  PE = 3 si la page est sur disque
  sinon, PE ≥ PF.
- le numéro PTPV de la page physique dans laquelle l'article couplé à la page virtuelle est chargé. Ce champ, superflu théoriquement, nous est imposé par l'instruction de chargement de la mémoire topographique.

On peut maintenant décrire le mécanisme d'accès à un article par le schéma suivant :



Mécanisme d'accès à l'information

N.B. On n'a pas figuré ici les mécanismes de détection de première écriture, ou de détection d'erreur de protection, qui utilisent également le déroutement.

Avant de pouvoir référencer un article de fichier, il faut l'associer à une page virtuelle. La primitive COUPLER (numéro de page virtuelle v, numéro de fichier F, numéro d'article) assure cette association en remplissant l'entrée v dans la TPV avec le numéro de fichier F, le numéro d'article, la protection PF et la valeur 3 pour PE. Un couplage n'implique aucun chargement de l'article concerné. Ce n'est que lors du premier accès que la protection PE joue et que le transfert est éventuellement demandé.

Quand un fichier est partagé, il ne lui correspond qu'une seule entrée dans la TFO. Un seul exemplaire des articles utilisés figure donc en mémoire centrale. De même un article de fichier peut être couplé à plusieurs pages virtuelles d'une même mémoire virtuelle.

Pour que des usagers puissent partager une procédure réentrante, dont une seule copie existe, il suffit que cette procédure soit couplée aux mêmes adresses virtuelles dans les différentes mémoires virtuelles .

Si le couplage est utilisé pour associer des pages virtuelles et des articles contenant un programme exécutable, il est clair qu'on peut définir un recouvrement ("overlay") au moment de l'exécution. Il suffit pour cela de coupler successivement les mêmes pages virtuelles à des articles différents.

Il pourrait sembler que le couplage introduise une structure de fichier très rigide. En fait, il faut le considérer plutôt comme une infrastructure facilitant les transferts entre mémoire principale et disques RAD. Il est possible d'utiliser cette infrastructure telle qu'elle est ou d'écrire des sous-programmes spécialisés, fonctionnant en virtuel et permettant au programmeur de manipuler des structures aussi compliquées qu'il le désire. Par exemple si on veut écrire un flot de n caractères (le premier étant à l'adresse virtuelle d'octet u) dans le fichier F, on utilisera le sous programme suivant :

- soit α l'article courant du fichier F. Il est couplé à la page virtuelle v. Soit r un index de remplissage ($0 \leq r < 2047$) qui pointe vers le premier caractère non écrit de l'article α .
- soit transfert (n, a, b) une procédure qui déplace n octets de l'adresse virtuelle a à l'adresse virtuelle b.

Le sous programme s'écrit :

```
E : begin
    if r + n < 2048 then
        begin   comment : l'article ∝ ne sera pas rempli ;
        transfert (n, u, 2048 * v + r) ;
        r :=  r + n
        end
    else
        begin   comment : l'article ∝ sera rempli, il faut alors
                          coupler un autre article ;
        transfert (2048-r, u, 2048 * v + r) ;
        n := n-(2048-r) ;
        ∝ := ∝ + 1 ;
        coupler (v, ∝ , F) ;
        u := u + (2048-r) ;
        r := o ;
        goto E
        end
    end
```

Lorsqu'on écrit dans une page virtuelle, l'article de fichier qui lui est couplé est modifié. Si on veut conserver une copie intacte d'un fichier, on doit le recopier dans un autre fichier.

3.5 Fichiers

3.5.1 Généralités

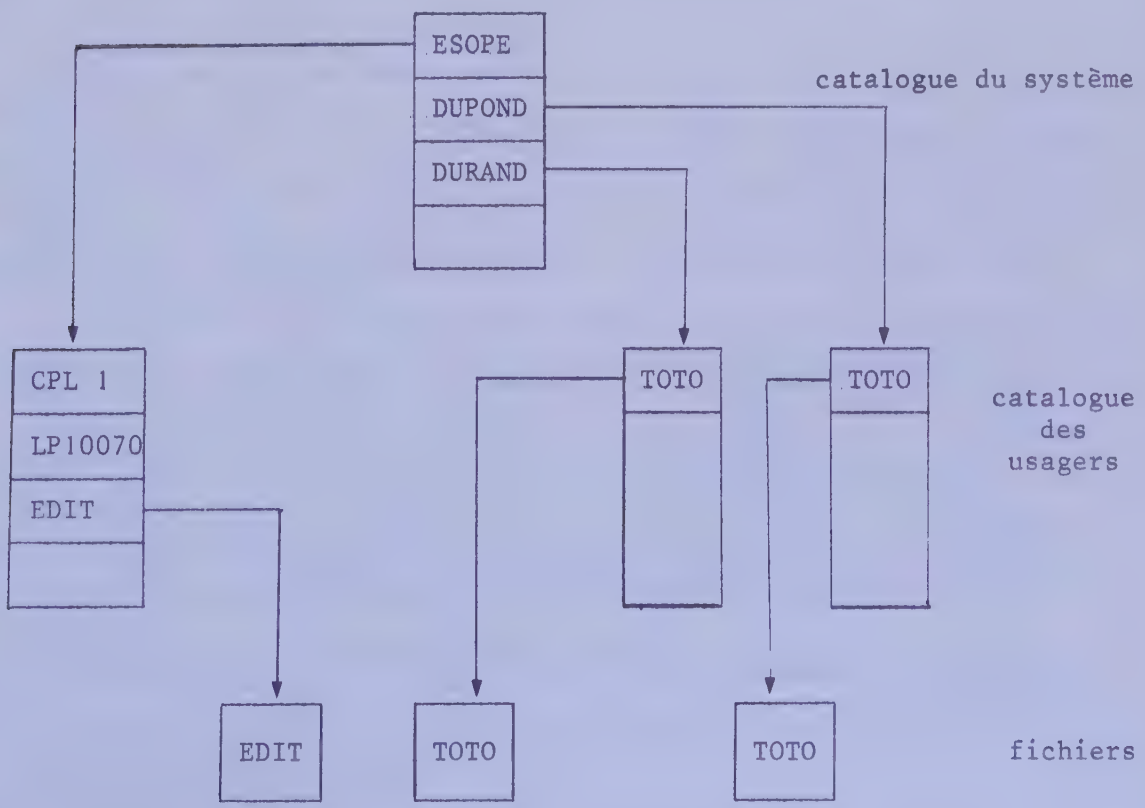
Dans un système à accès multiple, la gestion des fichiers joue un rôle plus important que dans un système classique. En effet, en plus des fonctions habituelles d'un système de fichier, on lui demande, comme les entrées-sorties sur téléimprimeurs sont très lentes, de conserver automatiquement pendant un temps en principe indéfini, les programmes et les données des usagers. Les rôles du système de fichiers d'ESOPE sont les suivants :

- conserver les informations que les usagers n'ont pu mettre en mémoire principale, faute de place,
- permettre le partage d'information entre usagers,

- conserver d'une session à l'autre, les programmes et les données d'un client du système,
- servir de relais entre les usagers et les périphériques classiques,
- permettre la communication entre les usagers et le travail de fond.

Pour réaliser un système de fichiers capable de remplir ces rôles nous nous sommes fixés comme objectifs, la simplicité et la généralité de la conception, la facilité de réalisation et la sécurité. Dans le cadre d'une recherche, nous ne voulions pas entreprendre une réalisation volumineuse, mais d'une part fournir un outil suffisant aux utilisateurs d'ESOPE et d'autre part étudier toutes les implications du couplage.

Aussi, nous avons choisi pour les fichiers, une méthode d'adressage symbolique à deux niveaux. Un premier catalogue du système fournit l'adresse des catalogues privés de chaque client. Chacun de ceux-ci donne l'adresse des fichiers du client. Un nom de fichier est alors formé par la concaténation de deux champm : un champ nom de client servant à l'entrée dans le premier catalogue et un champ nom de fichier servant à l'entrée dans le catalogue du client sélectionné au premier niveau.



ESOPE.CPL1, DUPOND.TOTO, DURAND.TOTO sont des noms de fichier. Chaque client peut baptiser ses fichiers sans contrainte.

Chaque fichier est un ensemble ordonné d'articles où l'article est l'unité d'accès à l'information et à la taille d'une page.

L'accès aux fichiers peut se faire selon l'un des 3 modes permis par le mécanisme de protection des pages virtuelles : lecture seule, lecture et exécution, tous accès.

Tout client a accès à l'ensemble des fichiers de son catalogue et aux fichiers des autres clients qui l'en ont autorisé. A chaque fichier, on associe la liste des clients qui ont droit de le partager.

Un fichier est soit une ressource à un seul accès, soit une ressource à accès indéfini. C'est-à-dire que dans le premier cas, un seul client peut y accéder à un moment donné tandis que dans le second de nombreux clients peuvent l'utiliser simultanément. Le premier cas correspond aux fichiers d'un client autorisés en écriture et le second aux procédures partageables du système tel CPL1, LP10070 ou EDIT.

3.5.2 Fichiers internes, fichiers externes

Comme les 3 disques RAD ne fournissent que 4095 pages au total et qu'ils doivent déjà contenir le système, on serait très vite limité si on se contentait d'eux comme support des fichiers.Aussi, avons-nous prévu l'utilisation des disques DIMAS.Cette unité soulève de nouveaux problèmes car on ne peut l'utiliser par la méthode du couplage. En effet, celle-ci suppose que lorsqu'une page n'est pas en mémoire, elle y soit amenée lorsqu'on y fait référence (méthode de chargement de page à la demande). Pendant son transfert, le processus est bloqué, mais continue d'occuper de la place en mémoire principale. Comme la DIMAS comporte un bras mobile qui supporte les têtes de lecture, le déplacement de ce bras peut être long (jusqu'à 150 millisecondes). Le temps d'attente pour une page est alors prohibitif et dégrade les performances du système en augmentant le temps d'oisiveté de l'unité centrale. Par contre, cette mémoire est bien adaptée au transfert global d'un fichier puisque sa cadence de transfert est élevée.

Aussi, avons nous choisi d'installer deux types de fichiers dans le système :

- les fichiers externes dont le support est la DIMAS. Ces fichiers sont catalogués, mais aucune opération n'est possible sur eux directement. Pour s'en servir, il faut d'abort les faire changer de support et les amener sur RAD,
- les fichiers internes qui ont le RAD comme support et qui sont accessibles par couplage. Toutes les opérations sont possibles sur ces fichiers.

Pour cataloguer des fichiers, nous avions la possibilité de faire exister :

- deux catalogues séparés pour les fichiers externes et internes,
- un seul catalogue pour tous les fichiers d'un client.

Le catalogue des fichiers internes doit être sur support couplable pour des raisons de sécurité. En effet, le système doit fonctionner même en l'absence de DIMAS. Le catalogue des fichiers externes peut être sur DIMAS.

Le critère de choix entre les deux solutions ci-dessus est la place disponible sur RAD. Un catalogue contient, pour chaque fichier :

- le numéro interne F s'il existe (entrée dans la TFO),
- le nom du fichier,
- la taille du fichier, en pages,
- le droit d'accès,
- les indications concernant le partage (numéros des clients, accès).

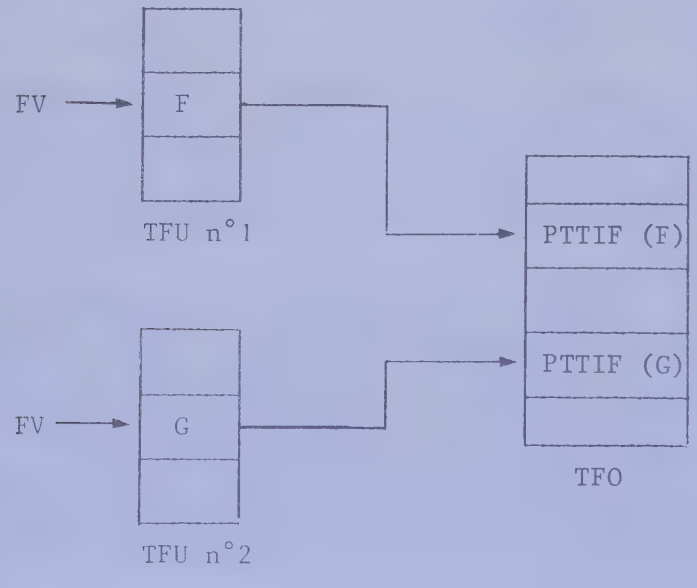
Au total, une dizaine de mots suffisent et l'ensemble du catalogue des fichiers d'un usager peut tenir sur une page si on se limite à 50 fichiers par usager.

Ceci permet de n'avoir qu'un seul catalogue pour tous les fichiers d'un client, solution la plus simple. On conserve sur DIMAS, à des emplacements fixes, les catalogues de tous les clients. Pour augmenter la rapidité d'accès au catalogue et éviter des attentes à chaque consultation, nous avons décidé que le catalogue d'un client X se trouverait placé momentanément sur RAD dans les cas suivants :

- X est un usager. Le client X est en cours de session à une console,
- ou bien, il existe un usager Y qui a annoncé son intention d'utiliser les fichiers du client X,
- ou bien, il existe un fichier de X pour lequel une entrée-sortie a été demandée à l'usager PTT, mais n'est pas encore terminée.

3.5.3 Opérations sur les fichiers

Les opérations sur les fichiers mettent en jeu la TFO, les TIF et des tables supplémentaires, les tables des fichiers d'un usager (TFU). A chaque usager est associé une TFU qui joue un rôle analogue à celui des "data control block" des systèmes de fichier batch (cas des moniteurs BPM et SIRIS 7 de la 10070 - cas du système IBM-OS-360). Chaque entrée FV de la TFU constitue un numéro de fichier virtuel, numéro qui doit être utilisé dans un programme au lieu du nom de fichier, si on veut que le programme soit réentrant ou utilisable avec des fichiers différents. La TFU est à une place fixe en mémoire virtuelle et contient un nombre maximum fixe d'entrées. A la compilation, les opérations sur les fichiers susceptibles d'apparaître dans un programme ont comme paramètre des entrées FV de la TFU, qui sont utilisées comme une variable "pointeur". Ces pointeurs ne reçoivent pas de valeur à la compilation. Ils n'en reçoivent une qu'avant l'exécution. Cette valeur est l'indice F du fichier interne qui aura été associé au fichier virtuel. En résumé, la TFU est une table d'indirection qui n'est remplie qu'à l'exécution et qui permet d'écrire des programmes de manipulation de fichier qui soient réentrants. La figure ci-dessous montre les TFU de deux usagers qui utilisent le même programme (donc même FV) avec des fichiers différents.



Les opérations sur les fichiers internes sont les suivantes :

CREER (FV) : Un fichier interne est créé. Il est repéré par FV. A partir de sa création et jusqu'à sa destruction, le fichier existe dans le système, on a accès à son contenu par l'intermédiaire du numéro virtuel FV. Au cours de cette opération :
- on crée une entrée dans la TFO, ce qui donne un F
- on réserve de la place sur disque RAD et on constitue la TIF
- on remplit l'emplacement FV de la TFU avec la valeur F.

DETRUIRE (FV) : C'est supprimer l'entrée correspondante dans la TFU, ôter si elle existe, l'entrée correspondante dans le catalogue, ôter l'entrée dans la TFO et libérer la place occupée sur disque RAD.

ASSOCIER (FV nom de fichier interne) : C'est remplir l'entrée FV de la TFU avec la valeur F qu'on trouve associée au nom dans le catalogue. Si FV était déjà associée à un fichier, celui-ci est dissocié.

CATALOGUER (FV nom de fichier) : La primitive créer constitue un fichier qui n'est pas répertorié dans le catalogue. On peut s'en servir puisque les primitives utilisent le paramètre FV. La primitive cataloguer permet de le conserver au-delà de la fin de la session de travail (à ce moment la TFU est détruite) et de lui donner un nom qui est rangé dans le catalogue du client.

OUVRIR (FV) : C'est demander d'accéder au contenu de FV en spécifiant un mode d'accès. Pour cela :
- on vérifie dans la TFO, que le fichier F n'est pas déjà ouvert, ou alors qu'il est partageable,
- on indique dans la TFU que l'entrée FV est associée à un fichier ouvert. Cette indication sert à l'allocateur de mémoire. On en verra l'emploi plus tard,
- on vérifie que le mode d'accès est permis.

FERMER (FV) : C'est supprimer jusqu'à la prochaine ouverture toute possibilité d'accès aux articles. Au cours de cette opération on doit :

- indiquer dans la TFU que l'entrée FV est associée à un fichier fermé,
- modifier la TPV et la mémoire topographique pour interdire tout nouvel accès à ce fichier. On recherche les entrées de la TPV qui contiennent le numéro F du fichier à fermer et on remplit les champs PE et PF correspondants avec la valeur 3 (tout accès interdit). Cette action est l'opération inverse du couplage.

On spécifie la longueur d'un fichier dans l'opération CREER. Il existe une primitive spéciale MODIFIER qui permet de changer les attributs d'un fichier et en particulier sa longueur.

Les opérations sur les fichiers externes sont :

RAPPELER un fichier externe, c'est le faire venir de la DIMAS vers le RAD. Pour cela, il faut :
- réserver de la place sur RAD pour y loger le fichier
- allouer une entrée F dans la TFO et construire la TIF correspondante

Cette primitive génère un appel à l'usager PTT qui fera le transfert d'information.

SAUVER un fichier interne consiste à le faire passer sur DIMAS. Ceci consiste à :
- lancer un appel au PTT pour qu'il fasse le transfert
- libérer la place que le fichier occupait sur le RAD
- libérer l'entrée dans la TFO et détruire la TIF.

SUPPRIMER un fichier externe, c'est oter l'entrée correspondante dans le catalogue et libérer la place occupée sur DIMAS.

Nous avons vu que les catalogues des clients se trouvaient sur DIMAS à des emplacements fixes. On ne transporte sur DIMAS que les catalogues comprenant des fichiers internes. Mais même dans ce cas, la version DIMAS du catalogue demeure, bien qu'elle ne soit pas mise à jour.

On utilise la présence des catalogues sur DIMAS pour assurer une sécurité plus grande au système.En cas de panne de la mémoire principale ou du RAD, on peut récupérer les fichiers sur DIMAS dans l'état où ils étaient avant l'opération RAPPELER à condition :
- de ne pas détruire le fichier sur DIMAS après l'opération RAPPELER, et de ne pas perdre son adresse,

- de ne pas écraser l'ancien fichier externe pendant l'opération SAUVER, mais au contraire d'écrire sur un nouvel endroit de la DIMAS. Remarquons au passage que cette méthode résoud le problème du retour sur DIMAS d'un fichier qui a grandi sur RAD. L'ancien fichier externe n'est détruit et sa place libérée que lorsque le nouveau a été écrit.

3.5.4 Fichiers de travail

Les programmes qui s'exécutent sont formés de deux zones, une zone procédure (protégée contre l'écriture) et zone de travail. La zone de travail contient les variables de travail de la procédure. Quand la procédure est partagée, elle est contenue dans un fichier partageable. Par contre, la zone de travail est toujours propre à un usager. A cause du couplage, il faut associer un fichier à cette zone de travail : c'est le fichier de travail.

Les fichiers de travail sont des fichiers internes (ayant donc une entrée dans la TFO) qui ne sont pas catalogués. Ils sont manipulés uniquement par leur entrée FV dans la TFU. Aussi, pour éviter de perdre tout accès à ce fichier, on interdit de dissocier un fichier de travail.(par exemple dans la primitive ASSOCIER)

Un fichier de travail peut être conservé et devenir un fichier permanent en le catalogant avec la primitive CATALOGUER. En fin de session, tous les fichiers de travail sont détruits.

3.6. Bibliographie

[1] CORBATO et al. The compatible time sharing system A programmer's guide MIT Press (1963)

[2] DALEY, DENNIS. Virtual memory, processes and sharing in Multics CACM 11,5 (mai 1968)

[3] WATSON. Time sharing system design concepts. Mac Graw Mill (1970)

[4] ANDERSON et al. Design of a time sharing system allowing interactive graphics. Proc. ACM nat. conf. (1968)

[5] BETOURNE, BOULENGER, FERRIE, KAISER, KRAKOWIAK, MOSSIERE. Notion d'espace virtuel dans le système Esope. Congrès AFCET (1970)

[6] CORBATO, VYSSOTSKY. Introduction and overwew of the Multics system. AFIPS vol 27 (FJCC 1965)

CHAPITRE 4

L'ATTRIBUTION DES RESSOURCES

4.1 Position du problème

4.1.1 Introduction

Les problèmes étudiés jusqu'ici étaient de nature intrinsèque. En particulier on a supposé que les processus se déroulaient sur un prccesseur virtuel en utilisant leur mémoire virtuelle : les ressources n'étaient pas limitées. Lorsqu'on fait entrer en considération la limitation des ressources physiques en passant des ressources virtuelles aux ressources réelles, le problème technologique de partage de ces ressources s'introduit.

La coopération des processus, l'accès à l'information mettent en jeu des mécanismes sur lesquels on a peu de prise. Une fois le choix de ceux-ci fait, les conséquences (coût en mémoire et surtout en temps d'unité centrale) sur les performances du système sont définitives. Comme ce sont des mécanismes structurels et qu'il n'y a aucune stratégie particulière d'emploi, aucun réglage n'est possible.

L'allocation des ressources, au contraire, utilise des stratégies diverses parmi lesquelles il faut choisir et qu'il est possible de paramétrer. Aussi de nombreux réglages sont-ils possibles. Il faut alors pouvoir définir très nettement les performances attendues du système et les mesurer. En fonction des résultats des mesures, il est ensuite possible de modifier le réglage des paramètres de façon à améliorer les performances obtenues.

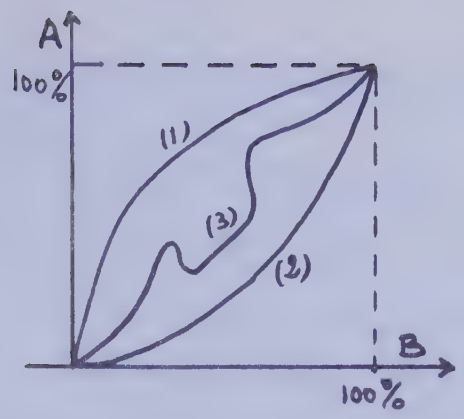
Dans le cas d'ESOPE, système conversationnel, les services à fournir ont été définis dans la spécification du système. Il en découle une structure particulière. Dans le cadre de cette structure, la performance essentielle qui est attendue est le temps de réponse. Par temps de

réponse, on entend le délai qui s'écoule entre la frappe du dernier caractère entré au terminal conversationnel (frappe du 'retour chariot' dans le cas du téléimprimeur) et l'arrivée du premier caractère du message de sortie. (Dans le cas d'ESOPE, où les téléimprimeurs fonctionnent en full duplex, cela correspond au délai entre l'émission et la réception du 'retour chariot'). Le temps de réponse d'un système conversationnel ne doit pas dépasser 3 secondes lorsque le travail demandé est élémentaire. Au delà, l'utilisateur du système ressent une gène notable. En fonction de ce critère, le nombre de terminaux du système pourra être déterminé.

Les ressources à partager dans le cas d'ESOPE sont la mémoire principale, la mémoire secondaire (disques RAD et DIMAS) et le temps d'utilisation de l'unité centrale. Les périphériques lents sont attribués de manière permanente à l'usager PTT. Quant aux terminaux interactifs, chaque usager en reçoit un lors de son admission dans le système et le conserve jusqu'à la fin de la session. Il n'y a aucun partage. L'attribution, donc l'entrée de nouveaux usagers, se fait tant qu'il reste des terminaux disponibles. Dans le cas de l'usager temps réel, celui-ci peut demander au système de lui attribuer des terminaux interactifs en supplément. Ici encore cette demande n'est satisfaite que si des terminaux ne sont pas employés. C'est donc au client voulant travailler en mode temps réel de s'assurer qu'il y a suffisamment de terminaux disponibles.

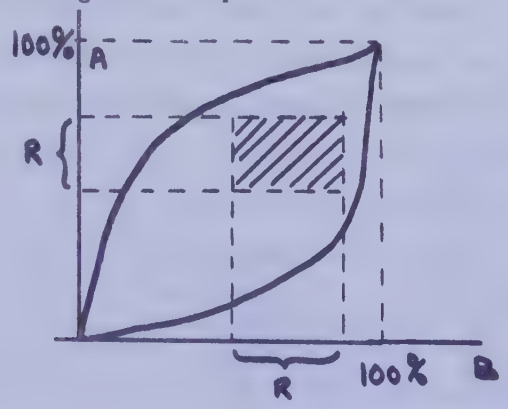
4.1.2 Etreinte fatale

Le partage des ressources rend possible un phénomène appelé blocage ou étreinte fatale, qui a pour conséquence de détruire le fonctionnement du système. Pour l'expliquer, représentons sur un axe les variations du taux d'exécution d'un processus. Ce taux est défini comme le rapport du nombre d'instructions exécutées au nombre d'instructions à exécuter. Dans ces conditions, l'exécution simultanée de deux processus A et B peut se représenter dans un plan, chaque axe de coordonnées représentant le taux d'exécution d'un processus. Comme le taux ne peut que croître lors d'une exécution réelle, les courbes réelles sont monotones, croissantes suivant l'un et l'autre axe.



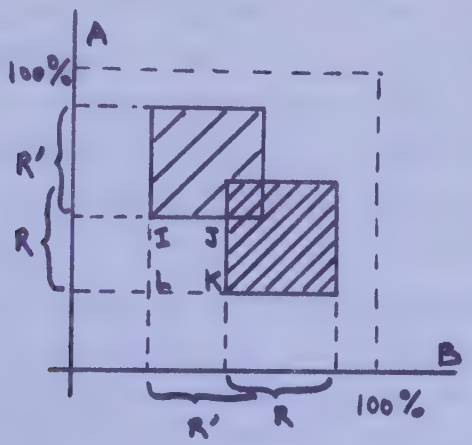
Aussi sur la figure ci-dessus, la courbe (3) ne représente rien de réel.

Supposons que pendant une phase de leur exécution, les deux processus aient besoin d'une ressource R à un seul point d'accès, c'est-à-dire une ressource utilisable par un seul processus à la fois. Les courbes représentant leur évolution ne peuvent traverser le rectangle hachuré sur la figure ci-après :



En effet, en un point situé à l'intérieur du rectangle, les deux processus utilisent simultanément la ressource R, ce qui est en contradiction avec l'hypothèse.

Considérons maintenant deux ressources R et R' en un seul point d'accès et supposons que chaque processus a besoin d'utiliser simultanément R et R' pendant une phase de son exécution.



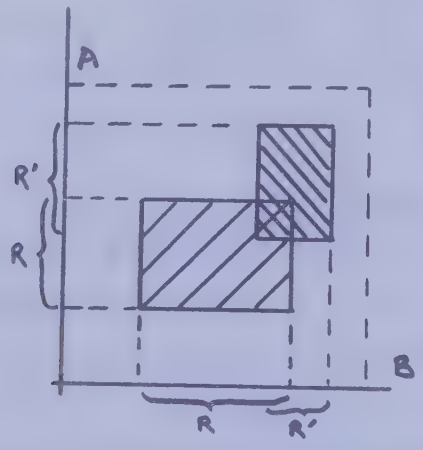
On constate dans le cas de figure ci-dessus que si une courbe représentative s'engage dans le rectangle IJKL, elle ne peut en sortir qu'en traversant une zone hachurée, ce qui est impossible. On dit qu'il y a blocage.

On notera que bien que le blocage ne soit effectif qu'au moment où la courbe atteint le coté IJ ou JK du rectangle, il est inévitable dès qu'on s'engage dans le rectangle, c'est-à-dire dès qu'il y a allocation de ressource. Toute technique de prévention du blocage devra donc à chaque allocation de ressource, garantir que la satisfaction de la demande n'entrainera pas de blocage.

Remarque : Dans le cas de figure précédent, A et B utilisent R et R' dans un ordre différent et le blocage traduit la situation où :

- A utilise R et a besoin de R' pour continuer
- B utilise R' et a besoin de R pour continuer.

Si A et B utilisent R et R' dans le même ordre, le blocage est impossible. C'est évident en considérant la figure suivante :



4.1.3 Interdépendance des allocations de mémoire et d'unité centrale

Dans les premiers systèmes à partage de ressources (CTSS [1]) , l'allocation de mémoire fut négligée car les auteurs pensaient que le problème critique était celui de l'allocation d'unité centrale, d'où le nom de système à partage de temps ("time sharing system"). Tandis que l'unité centrale était attribuée en fonction d'algorithmes compliqués, la mémoire était allouée selon une politique radicale :

- chargement au début de la tranche de temps allouée à un usager et vidage à la fin
- dans le cas d'ordinateurs possédant des canaux fonctionnant en parallèle avec l'unité centrale, partage de la mémoire en deux portions ; l'une d'elle contient le programme en cours d'exécution, l'autre fait l'objet d'un transfert d'entrée ou de sortie d'information.

Ces stratégies demandaient peu de connaissance sur les programmes à exécuter.

L'allocation de mémoire prit une grande importance à l'apparition des systèmes à pagination ([2, 3]) qui fonctionnent selon le schéma suivant :

- les programmes qui se trouvent au repos sur disque, ne sont plus chargés en totalité en mémoire avant leur exécution
- quand un programme fait référence à une page non encore chargée, il y a déroutement et demande de chargement de la page. C'est la stratégie de la page à la demande ("demand paging")
- l'emplacement physique où l'on va charger la nouvelle page (après avoir éventuellement vidé le contenu précédent) est déterminé par un algorithme spécialisé, dit algorithme de remplacement.

On notera que dans ces systèmes on ne limite ni le nombre des programmes fondés à se partager la mémoire, ni la place qu'un programme peut occuper à un instant donné. On notera aussi que l'algorithme de remplacement peut retirer des pages à un programme lui-même en attente de page, avec les conséquences que cela implique.

Ces systèmes ont donné lieu à de nombreuses expérimentations ([4]). La conclusion essentielle qui en a été tirée est que les performances du système (temps de réponse notamment) ne se dégradent pas lentement au fur et à mesure que la charge augmente, mais qu'on assiste à un écroulement brutal ("thrashing") lorsque la charge atteint un certain seuil. Ceci semble dû au fait que l'on peut voler des pages à un programme lui-même en attente de page et que le trafic sur les canaux d'entrée-sortie augmente beaucoup de ce fait et va jusqu'à saturer ces canaux.

Ceci a fait ressortir ce qui est considéré maintenant comme le point critique des systèmes à partage de ressource : le partage de la mémoire. Il est aussi apparu nécessaire de mieux connaître comment les programmes utilisent cette ressource au cours du temps. Si la répartition de la mémoire en tient compte, l'utilisation de la mémoire distribuée se fait dans de meilleures conditions.

On se trouve ainsi en présence de deux ressources :

- l'unité centrale pour laquelle la politique d'allocation peut être simple puisque ce n'est pas la ressource la plus rare
- la mémoire qui au contraire est la ressource la plus coûteuse et qu'il faut donc répartir le plus judicieusement possible.

L'ordre dans lequel est fait l'allocation de ces deux

ressources n'est pas indifférent. On peut noter qu'un changement d'allocation de l'unité centrale est relativement peu coûteux et qu'il demande quelques dizaines de microsecondes d'unité centrale (rangement-chargement des registres généraux et du mot d'état de programme). En revanche, tout changement de l'allocation de mémoire est coûteux car il fait intervenir des échanges avec la mémoire secondaire. Le temps d'occupation de l'unité d'échange peut atteindre plusieurs dizaine de millisecondes.

L'allocation d'unité centrale doit donc être subordonnée à l'allocation de mémoire. On choisit d'abord parmi les programmes prêts ceux qui vont se partager la mémoire pendant un temps donné, puis on alloue ensuite l'unité centrale à ceux-ci. De plus on verra que cet ordre immuable dans l'allocation de ressources de natures différentes permet d'éviter l'étreinte fatale.

4.2 Prévention du blocage [5, 6]

4.2.1 Introduction

Nous avons vu le cas classique de blocage, celui où deux ressources à un seul point d'accès A et B sont disponibles et où un processus P1 dispose de A et a besoin de B pour poursuivre, tandis que le processus P2 dispose de B et a besoin de A.

Un autre exemple de blocage est le suivant : soit un disque piloté par un processus D. D peut être considéré comme une ressource à laquelle d'autres processus demandent des services. Supposons que D a besoin d'une zone de mémoire Ƶ pour s'exécuter. Si un processus P occupe toute la mémoire et a besoin de faire appel à D avant de poursuivre, il y a encore blocage. Celui-ci est engendré par le fait qu'une demande de ressource entraine une autre demande. Une manière correcte de faire consisterait à fournir d'abord la zone de mémoire Ƶ, puis à activer D.

L'allocation des ressources doit donc éviter le blocage. Pour cela on se propose d'étudier les points suivants :

1) Etant donné un état quelconque du système, peut-on dire si le risque de blocage existe ?
2) Partant d'un état dans lequel il n'y a aucun risque de blocage, dans quelles conditions peut-on allouer des ressources à un processus P sans introduire de risque de blocage ?

3) Peut-on accepter un nouveau processus demandeur de ressources sans introduire de risque de blocage ?

4) Que se passe-t-il quand le nombre des ressources disponibles varie ?

Pour cette étude, nous nous plaçons avec les hypothèses suivantes :

1) Chaque processus annonce au départ quels seront ses besoins maximaux

2) Dans cette limite fixée au départ, tout processus demande et restitue des ressources à sa guise. Il s'engage toutefois à rendre toutes ses ressources au bout d'un temps fini.

3) l'allocateur a le droit de retarder toute demande mais il s'engage à la servir au bout d'un temps fini (mais non borné). L'allocateur ne peut forcer un processus à rendre des ressources.

4.2.2 Cas d'une seule classe de ressources

Le système contient K exemplaires d'une même ressource. Quand un processus fait une demande d'allocation, c'est pour p exemplaires de la ressource, quels qu'ils soient.

Soit n processus P1, P2,... Pn en compétition pour l'utilisation des ressources. Considérons tout d'abord K et n fixes.

Etat du système. L'état du système, à l'instant t, est décrit complètement par le nombre K et par deux vecteurs :

- D = (d1, d2,...dn). di est le nombre maximum de ressources dont le processus Pi aura besoin simultanément. D est une constante fixée en t = 0.

- A(t) = (a1(t),a2(t),...,an(t)). ai(t) est le nombre de ressources allouées au processus Pi à l'instant t. A(t) est modifié à chaque allocation ou libération de ressources.

Bien que le temps physique ait une variation continue, on ne considère que les instants t0, t1,...tq où des changements de l'état du système se produisent. La variable t peut dès lors être remplacée par une variable entière T, initialement à zéro et progressant de un à chaque changement d'état.

Un état (K, D, A(t)) du système est dit réalisable si les trois relations suivantes sont satisfaites :

R1 : $\forall i,\ 1 \leq i \leq n,\ di \leq K$

La demande maximale d'un processus est inférieure ou égale au nombre des ressources disponibles dans le système.

R2 : $\forall i, 1 \leqslant i \leqslant n, \forall t, ai(t) \leqslant di$

Un processus ne peut obtenir plus de ressources que le maximum qu'il a annoncé au départ.

Une demande qui satisfait R2 est appelée demande légitime.

R3 : $\forall t ; \sum_{i=1}^{n} ai(t) \leqslant K$

Le système ne peut allouer plus de ressources qu'il n'en dispose.

Si on pose $r(t) = K - \sum_{i=1}^{n} ai(t)$, on pourra utiliser indifféremment R3 ou R'3 :

R'3 : $\forall t, r(t) \geqslant 0$

r(t) indique le nombre des ressources disponibles à l'instant t.

La numérotation des processus est quelconque et les relations R1, R2, R3 ne dépendent pas de cette numérotation. Un état réalisable peut contenir le risque de blocage, comme le montrent les exemples suivants :

Exemples de blocage

1) K = 10, n = 2

D = (6,7), A(t) = (4,5)

Si les processus P1 et P2 demandent chacun 2 ressources, il y a blocage puisqu'il n'en reste qu'une de disponible

2) K = 10, n = 3

D = (2,6,8), A(t) = (1,3,5)

Une ressource est disponible ; on peut satisfaire P1. Une fois P1 terminé, il restera 2 ressources disponibles. Si P2 et P3 demandent chacun 3 ressources, il y a blocage.

Etats viables

Un état réalisable est dit viable, si quelles que soient les demandes légitimes des différents processus, le système peut les satisfaire au bout d'un temps fini.
Ainsi, dans le deuxième exemple ci-dessus, r(t) est assez grand pour satisfaire P1 qui peut donc se terminer et rendre toutes ses ressources. L'état serait viable si on pouvait continuer avec P2 ou P3 et ainsi de suite. Or, on ne peut servir la demande maximum ni de P2 ni de P3.

Dans le cas général, si le système est dans un état (al(t),...an(t)), cet état est viable si on peut trouver un processus Pk_1 tel que

$$dk_1 - ak_1(t) \leqslant r(t)$$

puis un processus Pk_2 (où $Pk_2 \neq Pk_1$) tel que

$$dk_2 - ak_2(t) \leqslant r(t) + ak_1(t)$$

etc...
puis un processus Pk_n (différent de tous les précédents) tel que

$$dk_n - ak_n(t) \leqslant r(t) + \sum_{i=1}^{n-1} ak_i(t)$$

Soit S l'ensemble des n processus rangés dans un certain ordre, on note S(k) le rang du processus Pk dans S. S est appelée une suite complète.

Par définition, un état réalisable (K,D, A(t)) est dit viable s'il existe une suite complète S telle que

R4 : $$\forall Pk \in S,\ dk \leqslant r(t) + \sum_{\ell : S(\ell) \leqslant S(k)} a\ell(t)$$

Comme $r(t) = K - \sum_{\ell=1}^{n} a\ell(t)$, on peut utiliser indifféremment R4 ou

R'4 : $$\forall Pk \in S,\ dk_1 \leqslant K - \sum_{\ell : S(\ell) > S(k)} a\ell(t)$$

Une suite complète vérifiant R4 ou R'4 est appelée suite viable.

On se propose de montrer que lorsqu'un état est viable, le système ne court aucun risque de blocage, c'est-à-dire qu'il existe au moins une politique d'allocation permettant de satisfaire dans un délai fini toute demande légitime des divers processus.

Comme le nombre des demandes légitimes est infini, on va d'abord définir les demandes les plus défavorables pour le système, puis prouver que, si le système se trouve dans un état viable, il existe une politique d'allocation permettant de traiter le cas le plus défavorable.

Du point de vue de l'allocateur, le cas le plus défavorable est celui où chaque processus utilise effectivement le nombre maximal de ressources qu'il a prévu au départ et où il ne rend aucune ressource avant d'avoir obtenu ce maximum.

Ce cas le plus défavorable sera traité si, partant d'un état réalisable A(to) on peut trouver une suite A(to), A(tl),...A(tn) d'états telle que :

- pour chaque état (K,D,A (ti)) est réalisable
- pour chaque processus Pk, il existe un instant tk tel que $ak(tk) = dk$
- ak est une fonction non décroissante de t sur (to, tk).

Théorème 1. Il existe une stratégie d'allocation permettant de satisfaire le cas le plus défavorable si et seulement si l'état initial du système est viable.

Démonstration :

a) Supposons viable l'état du système à to. Soit S une suite viable. Réordonnons les éléments de D et de A dans l'ordre de S.

$$dk \leq K - \sum_{i=k+1}^{n} ai(to) \qquad \forall k,\ 1 \leq k \leq n$$

Définissons une suite d'états A(tl),...,A(tk),...,A(tn) par

A(tk) : $a\ell(tk) = 0$

$ak(tk) = dk$

$a\ell(tk) = a\ell(to)$, $\ell < k$

Ces 3 relations traduisent les hypothèses suivantes :

- les processus de numéro inférieur à k ont déjà utilisé toutes leurs ressources et les ont rendues.
- le processus Pk demande toutes ses ressources.
- les processus d'indice supérieur à k n'ont rien demandé depuis to.

Alors,

$$\sum_{\ell=1}^{n} a\ell(tk) = dk + \sum_{\ell > k} a\ell\ (to) \quad K, \text{ d'après R'4}$$

La condition R3 est satisfaite, donc les états (K,D,A(tk)) sont réalisables et la suite A(to), A(t1)...A(tn) permet de satisfaire le cas le plus défavorable.

b) Supposons que, partant d'un état A(to), le système prenne une suite d'états réalisables A(to), A(t1),...,A(tn) tels que, après d'éventuels réarrangements des éléments de D et de A on ait :

- $\forall k,\ 1 \leqslant k \leqslant n,$ $\qquad ak\ (tk) = dk$

- ak (tk) fonction non décroissante de t sur $[to,\ tk]$

Par la suite $a\ell\ (tk) \geqslant a\ell\ (to)$ pour $\ell \geqslant k$

Comme A(tk) est réalisable, R3 donne pour tout k, $1 \leqslant k \leqslant n$:

$$d_k \leqslant K - \sum_{\ell \neq k} a\ell(tk) \leqslant K - \sum_{\ell > k} a\ell(tk) \leqslant K - \sum_{\ell > k} a\ell\ (to)$$

R4 est donc vérifiée en to. L'état initial est viable.

Remarques : R1 - Si le système est dans un état viable, il n'est pas obligatoire de servir les divers processus dans l'ordre de S. Le théorème 1 assure simplement qu'en cas de rareté des ressources, les processus peuvent être servis, en adoptant cet ordre.

R2 - Quand un processus fait une demande de ressource, le système peut la satisfaire sans danger si l'état obtenu après allocation est aussi un état viable.

Il y a ainsi à priori n ! permutations à essayer, nombre qui peut être réduit grâce au théorème suivant :

Théorème 2. Si l'état du système est viable, et si une sous suite S vérifie R4, alors S peut être étendu de façon à former une suite viable.

Demonstration :

Par hypothèse, l'état (K,D,A(t)) est viable. Il existe donc une suite viable Q telle que :

$$\forall Pk \in Q \qquad dk \leq r\ (t) + \sum_{\ell : Q(\ell) \leq Q(k)} a\ell\,(t) \qquad (1)$$

Soit S la sous suite vérifiant R4 :

$$\forall Pk \in S \qquad dk \leq r\ (t) + \sum_{\ell : S(\ell) \leq S(k)} a\ell\,(t) \qquad (2)$$

Définissons alors une suite complète S' de la manière suivante :

a) $S'(k) = S(k) \qquad \forall Pk \in S$

b) $Q(\ell) \leq Q(k) \Longrightarrow S'(\ell) \leq S'(k) \quad \forall Pk,\ P\ell \in Q\text{-}S$

Pour tout $Pk \in Q\text{-}S$ on déduit de cette relation b) que

$$\sum_{Q(\ell) \leq Q(k)} a\ell\,(t) \leq \sum_{S'(\ell) \leq S'(k)} a\ell\,(t) \qquad (3)$$

car Pk est précédé dans S' par tous les processus qui le précèdent dans Q, et aussi par tous les processus qui forment la suite S.
La relation (1) entraine alors

$$\forall Pk \in Q\text{-}S \qquad dk \leq r(t) + \sum_{S'(\ell) \leq S'(k)} a\ell\,(t) \qquad (4)$$

De (2) et de (4), on déduit que S' est une suite viable.

Corollaire - Si une sous suite vérifiant R4 ne peut être étendue de manière à former une suite viable, l'état du système n'est pas viable.

Algorithme 1 - On peut déterminer si un état du système est viable de la façon suivante :

S1 - R4 est vérifié par une suite vide S,

S2 - Essayer d'étendre le suite S avec un processus $Pk \notin S$, de telle façon que la suite étendu vérifie R4.

S3 - Si l'extension est possible, ajouter Pk à S et aller en S2.

S4 - Si la suite S est complète l'état est viable.

Ceci peut se décrire comme suit où :

S^* désigne le sous ensemble des processus qui font partie de la sous suite vérifiant R4

$\overline{S}$ l'ensemble complémentaire de S

T l'ensemble des processus avec lesquels on va essayer d'étendre S.

$\emptyset$ l'ensemble vide.

$S := \emptyset$; $T := S^*$;

WHILE $T = \emptyset$ DO

BEGIN

i := indice d'un élément de T ; $T := T - \{Pi\}$;

IF $di - ai(t) \leqslant r(t) + \sum_{Pl \in S} al(t)$ THEN

BEGIN

$S := S \cup \{Pi\}$; $T := S^*$

END

END ;

ETAT-VIABLE := $(S^* = \emptyset)$;

Cet algorithme a été introduit par E.W. DIJKSTRA [7] sous le nom d'algorithme du banquier.

Transitions entre états

Dans la pratique, l'état initial du système est viable. Il suffit alors de déterminer si, partant d'un état viable, le nouvel état obtenu après allocation au client Pk est lui aussi viable.

Théorème 3 : Si un état viable est modifié par une allocation au processus Pk et si on peut trouver une suite quelconque (non nécessairement complète) contenant Pk et vérifiant R4, alors le nouvel état est viable.

Démonstration :

Soit (K,D,A(to)) un état viable et (K,D,A(t1)) l'état transformé par allocation au processus Pk. Soit S une sous suite contenant Pk et vérifiant R4 à l'instant t1. On a :

$$\forall Pi \in S \qquad d_1 \leqslant K - \sum_{\ell \,:\, S(\ell) > S(i)} a_\ell(t1)$$

or

$$\forall \ell,\ \ell \neq k \qquad a_\ell(t1) = a_\ell(to)$$

et

$$a_k(t1) > a_k(to)$$

donc

$$d_i \leqslant K - \sum_{S(\ell) > S(i)} a_\ell(to)$$

S vérifie aussi R4 à l'instant to

Comme le système est dans un état viable à to, S peut être étendue pour former une suite S' viable à to, d'après le théorème 2. Montrons que S' est aussi viable à t1 :

a) S vérifie R4 à t1, par hypothèse

b) S' vérifie R4 à to donc

$$\forall P\ell \in S' \qquad d_\ell \leqslant r(to) + \sum_{S'(i) \leqslant S'(\ell)} a_i(to)$$

Tout processus $P\ell \in S' - S$ est précédé par Pk dans S (car $Pk \in S$).

Soit δk la nouvelle allocation à Pk, alors :

$$r(t1) = r(to) - \delta k, \quad a_k(ti) = a_k(to) + \delta k$$

Il vient, pour $\forall P_\ell \in S' - S$,

$$r(t1) + \sum_{S'(i) \leqslant S'(\ell)} a_i(t1) = r(to) - \delta k + \sum_{\substack{S'(i) \leqslant S'(\ell) \\ i \neq k}} a_i(to) + a_k(to) + \delta k$$

$$= r(to) + \sum_{S'(i) \leqslant S'(\ell)} a_i(to)$$

donc $\forall P_\ell \in S' - S$, la relation R4 est satisfaite à l'instant t1.

S' est une suite viable à l'instant t1.

Algorithme 2 . L'algorithme 1 peut être modifié :

- si $P_k \in S$, l'algorithme est terminé, le nouvel état est viable
- toute tentative d'extension de S commence par Pk.

d'où :

```
S := ∅ ;  T := S* ;
WHILE   (Pk ∉ S) ∧ (T ≠ ∅)  DO
          BEGIN
          IF Pk ∉ T  THEN i := k  ELSE i := index d'un élément de T ;
          T := T - {Pi} ;
          IF  di - ai(t) ≤ r(t) + Σ (Pℓ ∈ S) aℓ(t)     THEN
                BEGIN
                S := S ∪ {Pi}  ;  T := S*
                END
          END ;
ETAT-VIABLE := (Pk ∈ S) ;
```

Evolution des processus

Soit un processus Pk qui rend d ressources au système ; dk est inchangé ; si ak(t) diminue de d, r(t) augmente de d. La relation R4 est encore vérifiée et la transition est viable. On notera que cette restitution de ressources peut avoir pour effet d'autoriser d'autres processus à se poursuivre.

Si Pk rend d ressources et diminue sa demande maximale dk, la relation R4 est encore vérifiée et le nouvel état est viables.

Si K le nombre des ressources diminue, r(t) diminue aussi et la relation R4 peut ne plus être satisfaite. L'état résultant peut ne plus être viable. Ainsi une panne mettant un appareil hors service peut conduire au blocage.

Si un processus Pk augmente sa demande dk, cela peut également conduire à un blocage, sauf si Pk a rendu auparavant toutes ses ressources (ak(t) = 0)

La création d'un nouveau processus Pn+1 s'accompagne d'une création d'une $(n+1)^e$ colonne dans D et A(t) :

$$\begin{cases} dn+1, \text{ demande maximale de Pn+1, avec nécessairement } dn+1 \leq K \\ an+1(t) = 0. \end{cases}$$

Cette opération conduit à un état viable. En effet, par définition de r(t) on a :

$$r(t) + \sum_{\ell \leq n} a^{\ell}(t) = K$$

alors $dn+1 \leq K$ devient, sachant que $an+1(t)=0$:

$$dn+1 \leq r(t) + \sum_{\ell \leq n} a^{\ell}(t) + an+1(t)$$

ce qui est la relation R4.

La destruction d'un processus se produit après qu'il ait rendu toutes ses ressources. C'est un cas particulier de restitution de ressource. On a vu que le nouvel état est viable. La destruction du processus Pk conduit à supprimer la colonne k dans D et A.

4.2.3 Extension au cas de plusieurs classes de ressources

Supposons maintenant qu'il existe m classes de ressources.

L'état du système à l'instant t est maintenant défini par un vecteur et deux matrices.

$$K = \begin{vmatrix} k1 \\ k2 \\ \vdots \\ km \end{vmatrix}$$ indique les ressources du système

k_i = nombre d'exemplaires de la ressource de type i (i = 1,2,...m)

$$D = \begin{vmatrix} d_{11}\ d_{12}\cdots d_{1n} \\ \vdots \\ d_{m1}\ d_{m2}\cdots d_{mn} \end{vmatrix} = (d1,d2,\ldots,dn)$$

décrit les demandes maximales déclarées par les processus

d_{ij} = nombre maximum de ressources de type i dont le processus Pj aura besoin simultanément

$$A(t) = \begin{vmatrix} a_{11}(t)\ a_{12}(t)\cdots a_{1n}(t) \\ \vdots \\ a_{m1}(t)\ a_{m2}(t)\quad a_{mn}(t) \end{vmatrix} = (a1(t), a2(t),\ldots,an(t))$$

donne l'état de l'allocation des ressources à l'instant t

$a_{ij}(t)$ = nombre de ressources de type i allouées au processus Pj.

Introduisons une relation d'ordre entre vecteurs d'une part et matrices d'autre part.

Soit $U,V \in R^m$;

$U \leqslant V$ signifie que $\forall i$, $ui \leqslant vi$;

$U < V$ signifie que $(U \leqslant V) \wedge (\exists i, ui < vi)$

Soit F = (F1,F2,...,Fn) et G = (G1,G2,...,Gn) des matrices mxn.

$F \leqslant G$ signifie que $\forall k$, $Fk \leqslant Gk$;

$F < G$ signifie que $(F \leqslant G) \wedge (\exists k, Fk < Gk)$

Les relations $\geqslant$ et $>$ sont définies de manière semblable.

En adoptant ces relations d'ordre et cette écriture, les définitions, les théorèmes et les démonstrations données dans le cas d'une seule ressource se conservent intégralement et les résultats obtenus se généralisent [6].

Exemple de blocage

$$K = \begin{pmatrix} 4 \\ 5 \end{pmatrix} , \quad D = \begin{pmatrix} 4 & 3 \\ 3 & 4 \end{pmatrix}$$

$$A(t') = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} , \quad A(t'') = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} , \quad A(t''') = \begin{pmatrix} 1 & 1 \\ 2 & 2 \end{pmatrix}$$

$$r(t') = \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \quad r(t'') = \begin{pmatrix} 1 \\ 2 \end{pmatrix} , \quad r(t''') = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

A(t") et A(t''') sont des successeurs possibles de l'état A(t'). Tous trois sont des états réalisables.

A t' il n'y a pas de danger de blocage. Il suffit de faire attendre P1 jusqu'à ce que P2 ait utilisé toutes les ressources qu'il a demandées.

A t" il y a danger de blocage : si P1 ou P2 demandent leur maximum, k(t") est insuffisant pour les satisfaire. De même en t'''.

Remarque. Dans le cas où il y a plusieurs classes de ressources, les algorithmes 1 ou 2 s'appliquent globalement sur toutes les ressources à la fois. Les appliquer séparément sur chaque classe de ressource serait une erreur.

Par exemple si $K = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$ et $D = \begin{pmatrix} 4 & 2 \\ 2 & 4 \end{pmatrix}$

$A(t) = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$ est viable, $A(t') = \begin{pmatrix} 2 & 1 \\ 0 & 2 \end{pmatrix}$ l' est aussi, $A(t'') = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$ n'est pas viable.

Cependant quand on considère successivement les deux classes de ressources :

K = 4, D = (4,2), A(t") = (2,1) est viable

K = 4 D = (2,4) A(t") = (1,2) est aussi viable.

4.2.4 Remarques

Condition suffisante :

Si la somme des demandes maximales d'une ressource est inférieure au nombre d'exemplaires de cette ressource

$\exists i \quad \sum_j d_{ij} \leq ki$

toute demande de ressource peut être satisfaite sans problème.

Cas de plusieurs demandes d'allocation

Si plusieurs processus ont demandé des ressources, le choix de celui pour lequel on va exécuter l'algorithme 2 (celui qui sera le processus Pk) est fait avant l'exécution de l'algorithme et en est indépendant. L'algorithme n'introduit aucune priorité entre les processus. C'est le rôle du module de gestion des processus.

Si ce module traite les processus selon l'ordre premier arrivé, premier servi (FIFO, first in first out) en associant une file à chaque classe de ressource, on peut tomber dans un cas de blocage artificiel, dû uniquement à cette politique de gestion. En effet : Soit

$K = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad D = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix} \quad A(t) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Supposons les événements suivants :

P2 demande R2, il est mis en attente

P1 demande R2, il est mis en attente derrière P2

P3 libère R2. Comme P2 est en tête, on essaie de le satisfaire, ce qui conduit à vérifier si on a un état viable $A(t') = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$. Or cet état n'est pas viable ; on refuse donc d'allouer R2 à P2.

Si la gestion des processus est telle qu'on arrête là les tentatives d'allocation, parce qu'on respecte l'ordre de la file, on se trouve placé en blocage artificiel [8]. Pour l'éviter, il suffit de poursuivre l'essai dans la file jusqu'à trouver une demande conduisant à un état viable ou jusqu'à la fin de la file.

Attente indéfinie [8]

Si au contraire, on impose de satisfaire toutes les demandes qui conduisent à des états viables, on peut aboutir à faire attendre indéfiniment certains processus.

Soit un système où K = 2, D = (1 1 2), A(0) = (1 0 0)
Si la séquence suivante est possible :

A(0) = (1 0 0)

A(1) = (1 1 0)

A(2) = (0 1 0)

A(3) = (1 1 0)

A(4) = A(0)

Le processus P3 ne reçoit jamais de ressource.

Pour que cette séquence puisse se réaliser, il suffit que P3 ait une priorité inférieure à P1 et à P2 et que ceux ci se coalisent de façon qu'ils ne libèrent jamais tous les deux leur ressource en même temps.

Cette séquence serait en contradiction avec l'hypothèse initiale numéro 2 qui stipule que tout processus s'engage à rendre ses ressources dans un temps fini. Elle est toutefois possible si P1 et P2, au lieu, d'être un processus unique sont une suite de processus qui naissent et meurent (par exemple, un flot d'entrée de travaux dans le système).

Il apparait donc qu'il faut être prudent dans le choix des algorithmes de gestion des processus.

Simplification

Havender [9] a montré dans le cas de l'allocation de mémoire et d'unité centrale pour un monoprocesseur $K = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ que l'on pouvait éviter le blocage en allouant successivement mémoire et unité centrale, toujours dans le même ordre.

On peut étendre ce résultat pour un système où $K = \begin{pmatrix} k1 \\ 1 \end{pmatrix}$. En effet quand on alloue la ressource R1 (mémoire), la ressource R2 (unité centrale) a été rendue au système et alors $r_2(t) = 1$. Dans ce cas, la relation R4 s'écrit

$$\forall P_k \in S,\ (d_{1k}(t) \leq r1(t) + \sum_{S(\ell) \lesssim S(k)} a_{1\ell}(t)) \wedge (d_{2k}(t) \leq r2(t) + \sum_{S(\ell) \lesssim S(k)} a_{2\ell}(t))$$

Or la seconde partie de la relation est toujours vérifiée puisque $d_{2k}(t) = 1$ et $r2(t) = 1$.

On est ramené au cas d'une seule ressource R1 avec k1 exemplaires.

4.2.5 Cas d'ESOPE

L'algorithme du banquier suppose que l'allocateur ne peut forcer un processus à rendre des ressources. On verra plus loin que la mémoire et l'unité centrale ne sont pas gérées selon cet algorithme parce qu'on peut obliger un usager à les rendre au système.

Par contre la table des processus et la tables des sémaphores qui sont deux ressources communes aux divers processus du système peuvent être régies par cet algorithme.

Un processus P fait une demande de place dans la table des processus lorsqu'il crée un nouveau processus (primitive créer). De la place y est libérée lorsqu'un processus meurt (primitives détruire ou disparaître)

Une demande de place dans la table des sémaphores est faite lorqu'un processus crée un nouveau sémaphore (primitive déclarer). La libération de place intervient à la disparition du sémaphore (primitive effacer).

Comme les primitives apparaissent dans les algorithmes des processus, il n'est pas possible au système de forcer un processus à rendre les sémaphores ou les processus qu'il à créés. C'est un cas où s'applique l'algorithme du banquier avec deux classes de ressources, processus et sémaphores.

4.3 Utilisation de la mémoire par les programmeurs

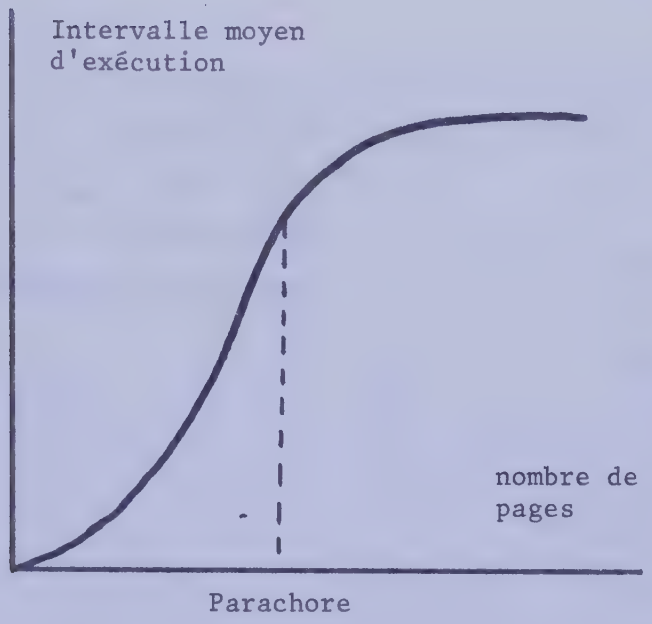
4.3.1 Comportement dynamique des programmes

On se propose d'étudier la succession des références à la mémoire au cours de l'exécution d'un programme. Comme nous nous plaçons dans le cas d'un système avec pagination, il suffit d'analyser la suite des pages référencées.

Notion de parachore

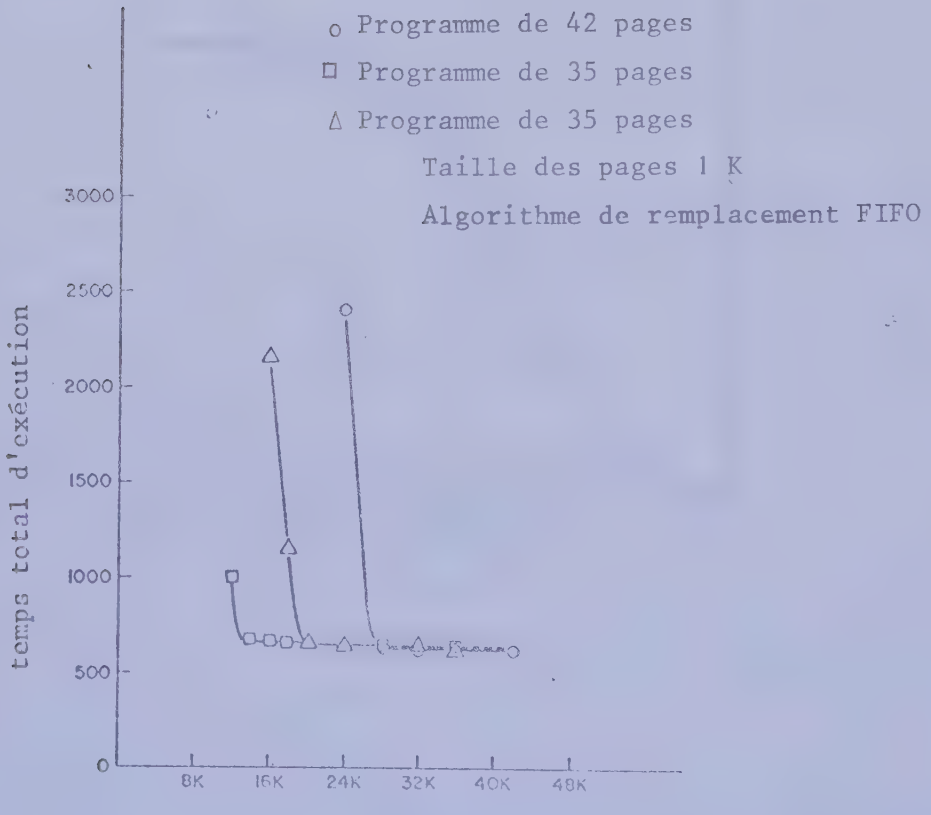
Soit un programme auquel on alloue une quantité s de mémoire, inférieure à la taille du programme. En s'exécutant, ce programme fera référence à des pages non chargées en mémoire. On aura une alternance d'intervalles d'exécution et de temps d'attente pour chargement de la mémoire.

Il est intéressant de connaitre l'intervalle d'exécution moyen τ qui sépare deux demandes de pages non chargées en fonction du nombre de pages allouées au programme. En toute rigueur τ est également fonction des caractéristiques du programme, de la taille des pages et de l'algorithme de remplacement choisi.
On obtient la courbe expérimentale ci-après : ([3]).



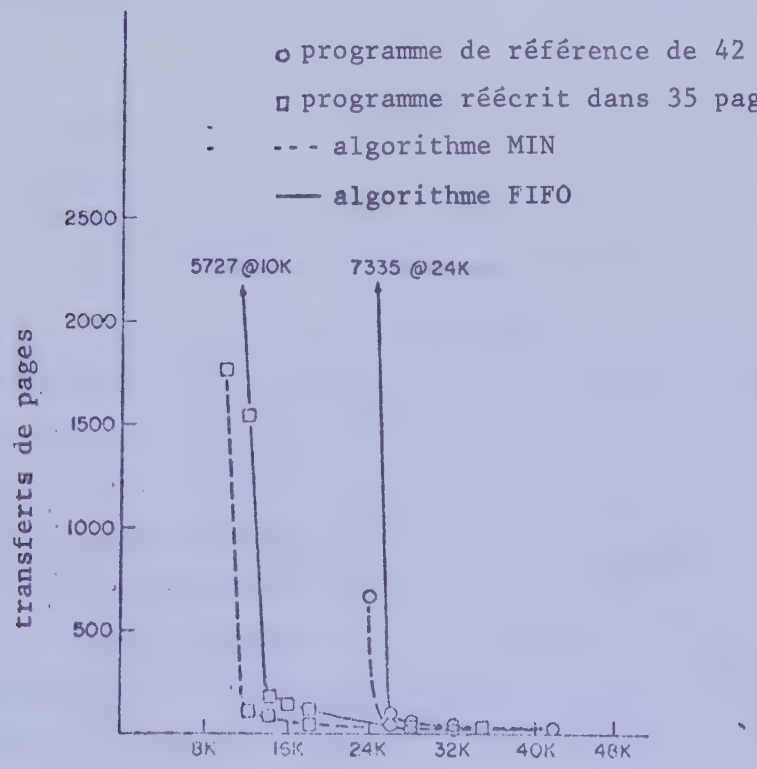
Cette courbe comporte un point appelé "parachore" en deça duquel les performances sont mauvaises car les défauts de page sont nombreux. Au delà du parachore par contre τ augmente peu lorsque le nombre des pages allouées augmente. Comme le parachore peut être très inférieur à la taille totale du programme, il est intéressant de n'allouer au programme qu'un nombre de page égal à ce parachore.

Brawn et Gustavson [10] ont obtenu des résultats expérimentaux analogues en mesurant le temps total d'exécution de programmes en fonction de la taille de mémoire allouée. On retrouve très nettement le parachore



Influence des algorithmes de remplacement

Brawn et Gustavson ont mesuré le nombre de défauts de page en fonction de la taille de mémoire allouée et de différents algorithmes de remplacement. L'algorithme MIN est l'algorithme donnant le minimum théorique de transferts vers la mémoire centrale. Il n'est pas utilisable dans les systèmes, car il prend en compte les accès futurs aux pages. Il est dû à Belady [11].



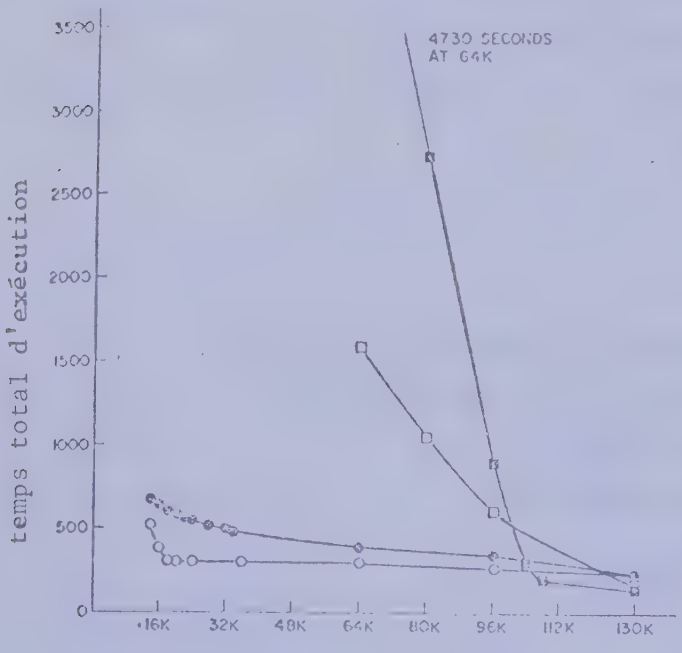
Taille de la mémoire allouée
(K = 1024 mots)

L'influence de ces algorithmes est faible.

Influence du style de programmation

Brawn et Gustavson ont comparé quatre versions d'un programme de tri. Les changements d'une version à l'autre n'affectaient

en rien le nombre total de pages référencées mais modifiaient le nombre des références à l'information. Les résultats obtenues sont spectaculaires et dépassent de très loin l'influence des algorithmes de remplacement.

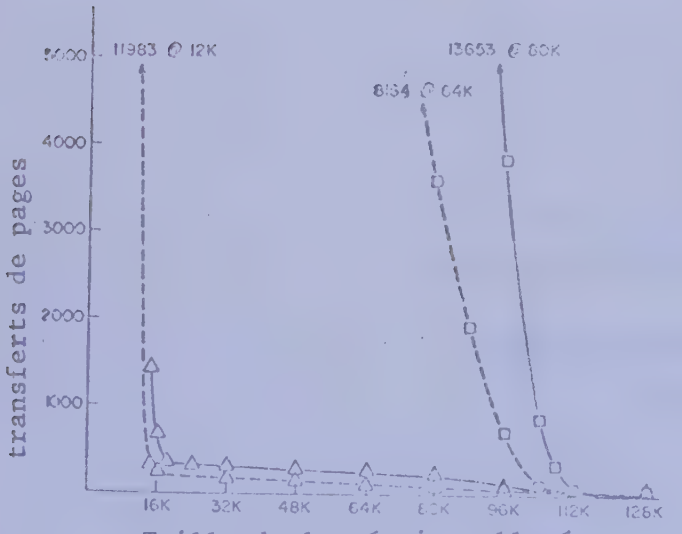


Taille de la mémoire allouée
(K = 1024 mots)

- programme de tri standart
- programme modifié 1
- programme modifié 2
- programme modifié 3

Taille des pages : 1 K

Taille des programmes : 129 pages



Taille de la mémoire allouée
(K = 1024 mots)

- programme standart
- programme modifié 3
- algorithme MIN
- algorithme FIFO

4.3.2 Modèle du Working Set [12]

Denning admet au départ que le comportement d'un programme est régi par le principe dit principe de localité : pendant

un certain intervalle de temps, un programme privilégie pour ses références un certain nombre de pages. La probabilité qu'un programme fasse référence à une page donnée n'est pas uniforme. Ce principe se justifie par les remarques suivantes :

- les instructions d'un programme sont rangées en séquence
- les programmes sont le plus souvent modulaires
- les données sont regroupées en vecteurs, tableaux ou sous ensembles
- les programmes contiennent des boucles.

Le "working set" $W(t,\tau)$ d'un programme P à l'instant t est l'ensemble des pages auxquelles le programme a fait référence dans l'intervalle $(t-\tau,t)$. La taille du working set $TW(t,\tau)$ est le nombre des pages référencées pendant $(t-\tau,t)$. Denning a montré que, pour τ petit, $W(t+\alpha, \tau)$ est voisin de $W(t,\tau)$.

Ces notions peuvent être utilisées dans un allocateur en n'autorisant un processus à utiliser l'unité centrale que si son working set $W(t,\tau)$ est en mémoire (on élimine ainsi le phénomène d'écroulement ou "thrashing") ou bien, dans une approximation grossière mais réalisable dans un système de page à la demande, que si un nombre de pages égal à la taille $TW(t, \tau)$ peut être réservé à ce processus.

Le choix de τ est important. Denning montre qu'il dépend du temps T de transfert d'une page en mémoire centrale et qu'un compromis convenable est de prendre τ de l'ordre de 2T.

4.3.3 Réalisations pratiques

La mesure en dynamique du parachore ou du working set est trop complexe , Aussi a-t-on réalisé seulement des méthodes approchées.

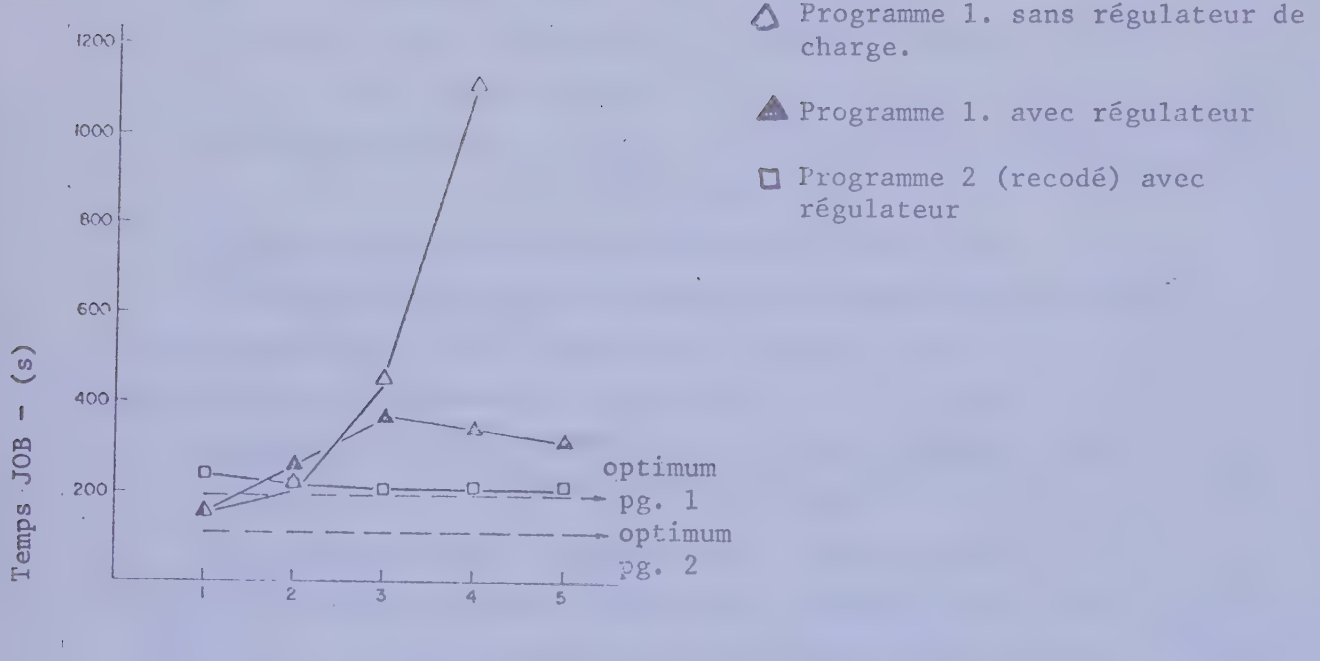
Régulateur de charge [13]

Soit un système en multiprogrammation dans lequel on évalue à tout instant le nombre des défauts de page et le taux d'oisiveté de l'unité centrale. Lorsqu'on constate que le nombre des défauts

de page est élevé alors que l'unité centrale est peu occupée, on retire un programme de la mémoire. Et inversement. Ceci a pour but de fournir aux programmes en mémoire une taille proche du parachore.

Ce système a été utilisé par Shils qui a mesuré, avec et sans régulateur de charge, le temps d'exécution de programmes en fonction du taux de multiprogrammation.

Les résultats sont donnés par les courbes suivantes



Catégorie

Chaque programme appartient à un moment donné à une certaine catégorie. A chaque catégorie correspond une taille de mémoire

et un temps de résidence en mémoire.

Soit NDISP le nombre des pages physiques disponibles à un instant donné. Un nouveau programme ne peut recevoir de mémoire que si NDISP est supérieur à la taille de catégorie. Dans ce cas on diminue NDISP de cette taille. Quand un programme s'exécute, il peut accumuler des pages physiques jusqu'à concurrence de la taille fixée par sa catégorie ou jusqu'à écoulement du temps de résidence. Lorsqu'une de ces limites est atteinte, le programme doit restituer l'unité centrale et toute la mémoire qu'il occupe. NDISP est augmenté en conséquence. Par ailleurs le programme change éventuellement de catégorie :

- s'il a chargé un nombre de page nettement inférieur à la taille de sa catégorie, on le mettra dans une catégorie où la taille est plus faible
- s'il a chargé la taille de sa catégorie bien avant la fin du temps de résidence, on le mettra dans une catégorie où la taille est plus forte

On constate que par ce jeu des catégories on peut :

- réserver de la place en mémoire pour un nombre de pages proche de la taille du working set, et donc éviter l'effondrement ('thrashing')
- éviter néanmoins de garder en mémoire des pages trop vieilles, en dehors du working set
- ajuster dynamiquement l'estimation du working set, en fonction du comportement du programme.

Cette notion de catégorie a été introduite dans le système EMAS [14] et est utilisée dans ESOPE.

4.4 Comportement des usagers conversationnels

En mode d'exploitation conversationnelle, ce qui attire l'usager c'est l'interaction directe avec le système, c'est à dire une réaction rapide à la commande ou à la donnée qu'il vient de taper sur sa machine à écrire. Pour concevoir le système et plus particulièrement l'allocateur de ressources de façon qu'ils rendent le meilleur service possible à chaque usager, il est indispensable d'avoir une idée du comportement de l'usager devant sa console, c'est à dire en particulier

de connaitre le temps moyen qu'il consacre à la réflexion et à l'écriture de ses messages.

Le cycle de l'activité d'un usager se déroule ainsi : il réfléchit à sa demande, tape son message qui est soit une commande au moniteur ou à un traducteur, soit une donnée, attend la réponse et examine le résultat. Puis le cycle recommence.

On appelle interaction l'intervalle de temps défini par deux messages consécutifs. Une interaction peut être divisée en une partie dépendant uniquement du système et une partie correspondant à l'activité de l'usager. La première partie, appelée temps de réponse comprend l'exécution proprement dite du message, après une attente dans une file, et la sortie d'une réponse. Le temps de réponse est un élément caractéristique du système et un des paramètres les plus utiles pour jauger ses performances. La partie usager ou temps de préparation débute une fois la réponse complètement sortie. L'utilisateur examine le résultat, réfléchit et tape le prochain message. Reflexion et écriture se déroulent en parallèle.

Tous ces intervalles de temps sont des grandeurs aléatoires qui fluctuent d'un individu à l'autre et qui pour un individu donné évoluent avec la nature du programme considéré. Par contre l'expérience montre [15] que le comportement moyen d'un usager est indépendant du nombre d'utilisateurs en conversation, de l'heure de travail (travail nocturne) et même du jour (travail de week-end)

Bryan a accumulé des statistiques sur 20 000 heures de travail à la console avec le système JOSS de la RAND. [16]
De son coté Scherr a analysé 80 000 commandes du système CTSS correspondant à 112 heures d'exploitation [15].
Les résultats suivants ont été obtenus :

	temps d'interaction		temps de réponse	
	moyen	median	moyen	médian
JOSS	34s	11s	10s	1,7s
CTSS	35s	11s	-	-

Pour chaque grandeur mesurée, on donne la valeur moyenne ainsi que la valeur médiane, c'est à dire la valeur qui, une fois sur deux, est dépassée. Il est important de noter que la moyenne donne une idée fausse du comportement d'un usager : en effet la distribution des grandeurs est très asymétrique si bien que la valeur moyenne est rarement observée tandis que la valeur médiane est plus fréquente en général.

Scherr montre également que la structure d'une interaction est très sensible à la nature de la tâche entreprise. Faire alors une moyenne des résultats sur la variété des tâches possibles est sans grande signification.

A cause de ces deux raisons, importance de la distribution des variables aléatoires ou corrélativement peu de signification de la moyenne, et sensibilité au type de travail, les statistiques données ne sont guère exploitables. En outre la définition de la moyenne n'y est pas très claire : on ne sait pas toujours sur quels ensembles on fait la moyenne.

Il est donc difficile de caractériser un usager en raison des très grandes fluctuations de structure des interactions. Par contre le comportement d'un ensemble d'usagers est à priori plus régulier, puisque tous les programmeurs ne sont pas simultanément en train de réfléchir ou de taper un message, si bien que le système voit arriver un flot de demandes de débit assez constant. On peut donc espérer schématiser plus aisément un groupe d'usagers qu'un seul. Denning [17] est parti de cette idée et a introduit la notion de console virtuelle, console qui n'a guère de rapport avec une console réelle, mais qui est telle qu'un ensemble de consoles virtuelles est supposé représenter un ensemble de consoles réelles. Grâce à ce modèle il est facile de déterminer la probabilité pour qu'un processus d'un usager soit bloqué en attente de message , de calculer combien de temps il restera bloqué en moyenne et aussi quelle est la taille des tampons des consoles.

4.5 Bibliographie

[1] CORBATO et al. The compatible time sharing system : a programmer's guide. MIT Press (1963)

[2] WILKES. Time sharing computer systems. Mac Donald (1968)

[3] KUEHNER, RANDELL. Demand paging in perspective. AFIPS vol 33 (FJCC 1968)

[4] DENNING. Thrashing : its causes and prevention. AFIPS vol 33 (FJCC 1968)

[5] HABERMANN. On the harmonious cooperation of abstract machines. These Math. dep. T.U., Eindhoven Pays Bas (1967)

[6] HABERMANN. Prevention of system deadlocks CACM 14,1 (janv. 1971)

[7] DIJKSTRA. Cooperating sequential processes, in Programming Languages, Genuys editor Academic Press (1968)

[8] HOLT. Comments on Prevention of system deadlocks. CACM 14,1 (janv 1971)

[9] HAVENDER. Avoiding deadlock in multitasking systems. IBM Syst. J.2,7 (1968)

[10] BRAWN, GUSTAVSON. Program behaviour in a paging environment. AFIPS V.33 (FJCC 1968)

[11] BELADY. A study of replacement algorithms for a virtual storage computer IBM Syst. J. 5,2 (1966)

[12] DENNING. The working set model for program behaviour CACM 11,5 (mai 1968)

[13] SHILS. The load leveller. IBM research RC 2233 (oct 1968)

[14] WHITFIELD. The organization of the university of Edinburgh time sharing system. International seminar Jerusalem 1968.

[15] SCHERR. An analysis of time shared computer systems, MAC TR 18, MIT (1967)

[16] BRYAN. JOSS : 2000 hours at a console, a statistical summary AFIPS vol 31 (FJCC 1967)

[17] DENNING. A statistical model for console behaviour in multiuser computers CACM 11,9 (sept 1968).

TROISIEME PARTIE

UTILISATION DES MECANISMES DE GESTION DU PARALLELISME DANS ESOPE

A la fin, nous dépendons des créatures que nous avons faites.

Goethe.

CHAPITRE 1

CONCEPTION ET REALISATION DE L'ALLOCATEUR

1.1 Principes de l'allocateur de ressources

1.1.1 Généralités

L'allocation doit être étudiée dans le cas où la charge du système est importante. Les différents programmes ne peuvent pas résider alors simultanément en mémoire centrale et une partie d'entre eux doit rester sur disque en attendant que de la place soit disponible. Le rôle de l'allocation de mémoire centrale est de sélectionner à chaque instant les programmes qui peuvent occuper de la mémoire et de déterminer les pages qui doivent être effectivement chargées.

Comme les canaux d'entrée-sortie peuvent fonctionner en simultanéité réelle avec l'unité centrale, on fait fonctionner le système en multiprogrammation.

Pendant que l'unité centrale exécute un programme, des pages sont chargées pour le compte d'autres programmes. Le rôle de l'allocation d'unité centrale est de choisir les programmes auquel donner l'unité centrale.

Pour réduire le temps d'oisiveté de l'unité centrale et augmenter la probabilité d'avoir à chaque instant un programme prêt à s'exécuter, on peut simplement chercher à augmenter le nombre des programmes partiellement résidents en mémoire. En contrepartie, on diminue la taille de mémoire utilisable par chaque programme et si la taille descend en dessous du parachore du programme, on augmente le trafic de pages. Un compromis est donc à trouver. Comme le débit des disques RAD est faible et ainsi que l'a prouvé la simulation que nous avons faite, tout notre effort doit porter sur la diminution du trafic de pages.

1.1.2 Principes de gestion

Les ressources partageables d'ESOPE sont : la mémoire principale, la mémoire secondaire et le temps d'utilisation de l'unité

centrale. Quelques idées ont guidé l'élaboration de la politique d'allocation des ressources :

- sécurité : éviter l'étreinte fatale
- efficacité : réduire le temps consacré à la gestion du système
- économie : favoriser le partage de ressources communes par plusieurs usagers (d'où des traducteurs réentrants)
- souplesse : allouer les ressources le plus tard possible. Ceci diminue aussi le coût d'utilisation des ressources, coût proportionnel à leur temps d'occupation
- adaptation : utiliser les renseignements connus à l'avance et les observations recueillies pour adapter la politique de l'allocateur.

Ces considérations nous ont conduit à énoncer pour l'allocation de ressource les principes de base suivants :

1er principe : La mémoire principale et le temps d'unité centrale sont partagés entre les usagers et non les processus. On espère ainsi tenir compte au mieux de l'existence de classes de processus indépendants et de l'utilisation de données communes dans chaque classe.

2e principe : la mémoire secondaire sur disques est partagée entre les fichiers. Ceci se fait sous contrôle du moniteur.

L'ordre dans lequel un usager reçoit la mémoire et l'unité centrale n'est pas indifférent. La première est fragmentée et longue à charger. La seconde est indivisible mais rapidement commutable. D'où la conséquence suivante.

3e principe : la mémoire principale est allouée avant l'unité centrale.

La mémoire du 10070 est divisée en pages. Cela permet de ne pas avoir tout le programme d'un usager en mémoire, mais de charger les pages à la demande et de bien distinguer deux niveaux dans la gestion de la mémoire :

- un niveau global où la mémoire disponible est répartie entre les usagers, à qui on réservé, en fonction de leur catégorie un certain nombre de pages (non identifiées individuellement). C'est la réservation.
- un niveau local où un usager utilise des pages physiques dans la limite du lot qui lui est imparti à la réservation. C'est l'allocation.

4e principe : La réservation précède toujours l'allocation qui se fait selon un service de page à la demande.

Remarquons que comme la mémoire et l'unité centrale sont toujours allouées dans le même ordre, il ne peut y avoir d'étreinte

fatale entre elles. Pour la mémoire, les mesures à prendre pour éviter le blocage sont prises au moment de la réservation.

5e principe : Une ressource ne peut être allouée à un usager pour un temps indéfini : il existe une durée limite d'allocation.

Ce principe découle de considérations évidentes de sécurité. Il résulte aussi de la volonté d'assurer un temps de réponse correct. Une ressource, mémoire ou unité centrale, n'est allouée à un usager que pour un temps déterminé au moment de l'allocation : pour la mémoire c'est le temps de résidence, pour l'unité centrale c'est le quantum.

1.1.3 Etats des Usagers

Nous avons vu qu'un processus avait deux états intrinsèques, actif et bloqué.

L'état d'un usager est déterminé par l'état de ses processus et par les ressources qui lui sont allouées :

- un usager est oisif si tous ses processus sont bloqués intrinsèquement
- un usager est prêt s'il est non oisif et si aucune ressource ne lui est réservée ou attribuée
- un usager est courant s'il est non oisif et s'il participe à la distribution des ressources.

On distingue trois types d'usagers courants :

- usager suspendu : il est en attente de pages dont il a demandé l'allocation,
- usager éligible : il attend la disponibilité de l'unité centrale. D'après le 3e principe, il a reçu les pages qu'il attendait.
- usager élu : il s'exécute car il dispose de toutes ses ressources.

Ces états se résument dans le tableau suivant :

Etat de l'usager	processus de l'usager	ressources allouées à l'usager	
		mémoire	UC
OISIF	tous bloqués	non	non
PRET	un actif, au moins	non	non
SUSPENDU	"	réservée, attente de page	non
ELIGIBLE	"	réservée, pas d'attente de page	non
ELU	"	"	oui

Nous avons autorisé un certain nombre seulement de transitions entre états.

Un usager oisif devient prêt dès qu'un de ses processus est débloqué.

Un usager prêt devient courant dès qu'on peut lui réserver un nombre de pages égal à la taille de sa catégorie.

Un usager quitte l'état courant dans trois cas :

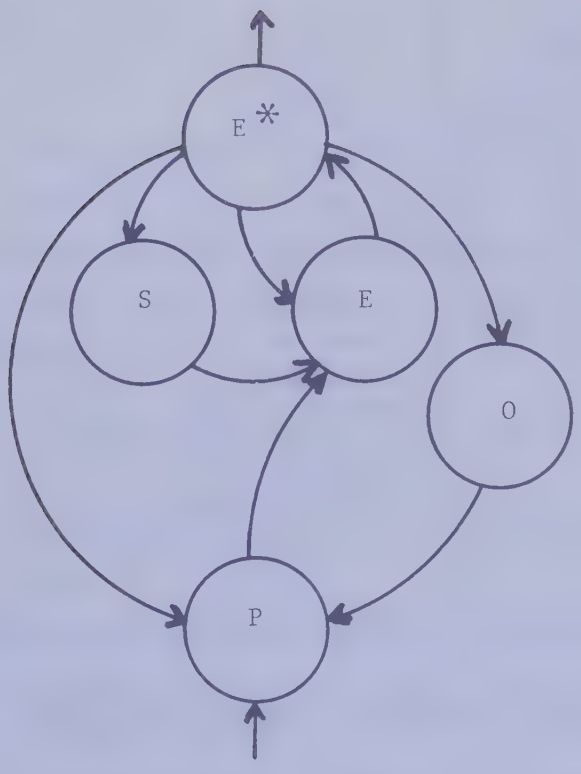
- si tous ses processus sont détruits, il quitte le système
- si tous ses processus sont bloqués, il passe dans l'état oisif
- si le système décide de lui retirer ses ressources (fin du temps de résidence ou tentative de dépassement de la taille associée à la catégorie), il passe dans l'état prêt.

A l'intérieur de l'état courant :

- l'usager élu libère l'unité centrale en cas de défaut de page (il passe alors suspendu) ou en fin de quantum (il passa éligible)
- un usager devient éligible dès que les pages qu'il a demandées sont présentes en mémoire centrale
- un usager éligible devient élu sur décision de l'algorithme d'allocation d'unité centrale.

Un usager qui entre dans le système le fait dans l'état prêt.

On peut récapituler les transitions sur le schéma suivant :



E* élu
E éligible
S suspendu
} C courant
O oisif
P prêt

1.2 Circulation des usagers

Nous allons analyser la circulation des usagers selon le schéma qui vient d'être donné. Nous étudierons d'abord les actions qui interviennent dans la boucle interne et qui concernent l'allocation de mémoire et d'unité centrale. Puis nous considérerons la boucle externe qui régit la réservation de place en mémoire.

1.2.1 La boucle interne de l'état courant

Cette boucle concerne les trois états élu, suspendu, éligible.

1.2.1.1 Etat Elu

Politique d'activation

Considérons l'usager élu. Par définition un de ses processus s'exécute sur l'unité centrale jusqu'à rencontrer un arrêt programmé (primitive P), ou un défaut de page. Comment va-t-on allouer l'unité centrale :

- à un processus actif d'un autre usager courant, pourvu que ce processus ne soit pas en attente de page ('activation simple') ?
- à un autre processus actif du même usager, s'il existe ('activation

multiple') ?

Remarquons que le passage de l'unité centrale d'un usager à un autre impose un chargement de la mémoire topographique et des clés de protection associées aux pages virtuelles tandis que le passage d'un processus à un autre d'un même usager demande un rangement du vecteur d'état (mot d'état de programme et registres généraux) de l'ancien processus et un chargement du vecteur d'état du nouveau processus. Dans l'activation simple, les deux opérations ci-dessus sont à faire, alors que dans l'activation multiple, la seconde seule intervient. Le temps passé dans le moniteur ("overhead") est donc plus faible dans ce dernier cas.

D'autre part, l'activation multiple exploite mieux le partage d'information entre processus coopérants. Supposons qu'un processus A ait amené en mémoire, par une succession de défauts de page, une zone de données qu'il partage avec un processus B du même usager, puisqu'il se bloque.

Si B est activé tout de suite après A, on utilisera le plus tôt possible une ressource immobilisée, qui permettra peut être de la libérer plus vite.

Enfin, l'activation multiple est conforme au choix d'allouer les ressources à l'usager et non aux processus.

Nous choisissons donc une politique d'activation multiple. On activera tous les processus actifs de l'usager élu, jusqu'à ce qu'ils soient tous bloqués ou en attente de page.

Conséquence pour le quantum

Le quantum d'unité centrale est alloué à un usager, globalement, et non à un processus.

Conséquence pour les demandes de pages

Quand un processus de l'usager élu tombe en défaut de page, on peut décider de demander immédiatement le transfert de cette page depuis le disque, puis d'activer un autre processus du même usager. Supposons que le temps de résidence de l'usager soit épuisé avant l'arrivée de la page en mémoire. Il faut alors libérer les pages d'un usager pour lequel un transfert est en cours. Cette opération est délicate, puisqu'elle demande l'arrêt d'une entrée-sortie. De plus le disque aura été utilisé inutilement et le système certainement freiné puisque le disque est une ressource critique. Nous rejetons cette façon

de faire et choisissons une méthode plus conforme d'ailleurs à la notion d'usager.

Quand un processus d'un usager fait une demande de page, on se contente de noter la demande dans une table réservée à cet effet, la table des pages demandées (TPD) et de bloquer ce processus derrière un sémaphore technologique propre à cet usager, S:REQ(USR). Dès que l'usager n'a plus de processus actif, et si son temps de résidence n'est pas épuisé, on demande le chargement de toutes les pages de la TPD. Dans le cas où l'usager doit quitter la mémoire, on conserve la TPD en vue d'une utilisation ultérieure. Remarquons que pour simplifier la gestion du temps, on ne teste le dépassement du temps de résidence qu'en fin de quantum ou que lorsque l'usager n'a plus de processus actif.

Conséquence pour les processus d'un usager.

Nous avons vu qu'il fallait pouvoir lancer successivement tous les processus actifs d'un usager et qu'il fallait aussi détecter le dernier processus d'un usager.

Pour activer successivement les processus d'un usager, on associe à chacun d'eux une priorité (elle est fixée par la primitive de création du processus) et les processus prêts d'un usager sont rangés, selon cette priorité dans une file propre à l'usager. Par processus prêt, on entend un processus actif qui n'est pas en attente de page. Dès qu'un processus est bloqué par une primitive P, il quitte la file prêt de l'usager pour rejoindre une file de sémaphore. Inversement tout processus libéré par une primitive V ou D quitte la file de sémaphore pour rejoindre la file des processus prêts de son usager. Quand le quantum d'unité centrale est actionné, il en résulte un changement d'usager élu on change par là même de file de processus prêts.

On pourrait détecter le dernier processus d'une file de processus prêts, en faisant un test dans la primitive P et dans la primitive de destruction de processus. Ce sont en effet ces deux primitives qui font élire un nouveau processus. Pour ne pas alourdir la primitive P qui est très souvent employée, il nous a semblé préférable d'introduire un processus supplémentaire par usager, le processus final. Ce processus a la priorité la plus faible de tous les processus de l'usager, il est donc toujours en queue de la file prêt de l'usager qui est ainsi facilement repérée. Le processus final ne doit jamais quitter la file prêt. Il se contente donc d'activer un processus du moniteur qui est chargé de faire quitter l'état élu à l'usager. Comme ce processus est plus prioritaire que final et qu'il a comme action de changer

l'usager élu, comme cet usager ne sera jamais réélu tant qu'il n'aura pas d'autre processus prêt que le processus final, le code de ce dernier peut être très simple et se limiter à :

```
FINAL : V(sémaphore du moniteur) ;
        goto FINAL ;
```

Attente prolongée

L'activation multiple peut entrainer comme conséquence de bloquer un processus pendant un temps très long et de fausser complètement certains raisonnements basés sur la priorité des processus.
Exemple 1. Soit en effet un processus A qui boucle.

Supposons que nous puissions, depuis le téléimprimeur de conversation par la touche "appel système", activer un processus plus prioritaire, appelé ici processus premier et que parmi ses fonctions ce processus premier puisse détruire ou arrêter le processus A. Supposons encore que le processus premier n'est pas résident et que dès qu'il est appelé il fasse un défaut de page. Que va-t-il se passer ?

a) - le processus A reçoit à nouveau l'unité centrale, qu'il garde jusqu'à la fin de quantum.
b) - comme l'usager passe de l'état élu à l'état éligible sans passer par l'état suspendu, la demande de page pour le processus premier n'est pas faite. Le processus A reçoit l'unité centrale quantum par quantum jusqu'à atteindre la fin du temps de résidence.
c) - l'usager quitte alors l'état courant pour passer dans l'état prêt en libérant toutes ses pages. Puis au bout d'un certain délai, il réintègre l'état courant. Supposons qu'il le fasse dans l'état éligible. Lorsqu'il devient élu, le processus premier est activé. Il refait alors son défaut de page, ce qui a pour conséquence d'activer A. Le processus A fait aussi un défaut de page et l'usager passe enfin suspendu. La page manquante du processus premier peut être appelée.

Remarquons que si pour fonctionner le processus premier a besoin de plus d'une page, il ne passe jamais, car la phase c) ne garantit le chargement que d'une seule page par processus.
De même, si le processus A boucle dans ses registres, il ne fait pas de défaut de page et le processus premier ne progresse jamais.

Supposons maintenant que nous ayons voulu être astucieux. A chaque défaut de page, avant de mettre la demande dans la TPD, on regarde si la page demandée n'est pas déjà en mémoire (on appelle cette

action, la récupération) et si c'est le cas, on relance le processus dérouté. Si dans notre exemple, le processus A peut récupérer les pages dans lesquelles il boucle, l'usager ne passe pas suspendu. Donc le processus premier ne progresse pas aussi longtemps que A peut faire de la récupération.

Pour pallier ce défaut, nous avons trois solutions possibles.

- rendre résident la partie du processus premier chargée de traiter l'appel système. Il n'y a plus d'attente après l'appel système. Le processus premier est activé dès que l'usager est élu.
- passer systématiquement par l'état suspendu en fin de quantum. Si le processus premier a besoin de n pages, l'attente maximum est le temps nécessaire pour fournir n quantum à l'usager.
- entrer dans l'état courant par l'état suspendu, en utilisant la TPD qui existait au passage de courant à prêt. Cette solution protège contre la récupération mais n'est valable que si le processus premier peut se contenter d'une page. L'attente maximum est le temps de résidence de l'usager, plus l'attente dans l'état prêt, plus l'attente dans l'état éligible.

Nous avons choisi la première solution pour ESOPE.

Exemple 2. Soit un processus A tombé en attente de page. Soit deux processus B et C qui se coalisent de telle sorte qu'ils ne soient pas bloqués tous les deux à la fois. La demande de page de A ne sera servie que lorsque B et C seront bloqués, et cela même si A est plus prioritaire qu'eux.

Ce cas peut se produire pour l'usager PTT où A, B, C sont des processus "facteurs" associés à des périphériques qui sont activés par les interruptions de fin de transfert. Remarquons toutefois que ce cas est très improbable car il suppose qu'un facteur soit un processus suffisamment long pour qu'un transfert vers un périphérique puisse se faire pendant qu'il s'exécute. Or, les facteurs d'ESOPE sont courts.

Traitement du défaut de page

Rappelons que la 10070 n'a pas de bit réservé pour dire si une page virtuelle est en mémoire physique ou non. Il nous faut utiliser le dispositif de protection virtuelle. De même on n'a rien pour dire si

une page a été écrite ou non. On utilise encore le dispositif de protection virtuelle.

En dehors des violations de protection virtuelle qui proviennent d'une erreur (accès à une page non couplée par exemple), deux cas peuvent se produire :

- 1er cas : une page non encore écrite est l'objet d'une tentative d'écriture. Il suffit de marquer cette page comme écrite aux endroits adéquats et de relancer le processus dérouté.
- 2ème cas : on fait référence à une page inaccessible. Il faut inscrire la demande de page dans la TPD. On y joint la nature de l'accès à la page. Si cet accès est une écriture, on pourra marquer la page comme écrite pour éviter un nouveau déroutement d'écriture. On bloque le processus dérouté derrière un sémaphore technologique propre à l'usager, S:REQ(usager).

Revenons un peu sur la notion de page inaccessible.Pour contrôler la réservation de mémoire, on verra que toutes les pages d'un usager doivent être marquées inaccessibles au moment où l'usager quitte l'état courant. Ceci permet d'utiliser le défaut de page pour compter les pages allouées. Sont donc inaccessibles, toutes les pages qui n'ont pas encore été allouées. Ces pages, couplées à des articles de fichier, se trouvent soit sur disque, soit en mémoire centrale. Il peut sembler dommage de ne pas utiliser ces dernières tout de suite. Si on veut le faire, il faut, dans le déroutement deuxième cas, localiser l'article du fichier couplé à la page et le récupérer s'il se trouve en mémoire centrale. Dans ce cas, on relance le processus dérouté.

Remarquons encore que la simple détermination du numéro de page virtuelle ayant provoqué le déroutement n'est pas triviale. En effet, le matériel ne délivre que l'adresse virtuelle de l'instruction au cours de laquelle il y a eu déroutement et il faut alors analyser successivement toutes les références à la mémoire faites par cette instruction, jusqu'à ce qu'on trouve la bonne. Or certaines instructions complexes peuvent engendrer des références à 5 pages différentes. Il serait très rentable de modifier le mécanisme du déroutement de façon à obtenir immédiatement, dans un mot d'adresse fixe, l'adresse virtuelle ayant provoqué la violation de protection.

1.2.1.2 Etat suspendu

Quand l'usager n'a plus de processus prêt, et que l'un au moins de ses processus est bloqué pour défaut de page, il essaie de passer dans l'état suspendu. (On considère le cas où l'usager reste

courant).

Préparation de la requête

On doit à partir de la TPD, préparer un ensemble de demandes au module de gestion des disques. Nous appelons requête cet ensemble de demandes.

Pour chaque page virtuelle notée dans la TPD, on doit :

- localiser l'article de fichier couplé à la page
- vérifier que cet article n'est pas demandé à une autre entrée de la TPD (cas de demande de deux processus à la même page ou à deux pages couplées au même article)
- récupérer l'article si il est en mémoire physique. Cette opération, qui économise un transfert, est obligatoire car la version sur disque associée à l'article peut être périmée.
- annuler la demande d'article si celle-ci a déjà été faite pour un autre usager
- dans le cas où l'article doit être transféré en mémoire centrale, allouer une page physique pour cette opération. Si cette page doit être vidée avant son utilisation, (cas des pages écrites, qui n'ont pas de copie sur RAD), on l'indique dans la requête.

Si après toutes ces actions, la requête est vide, l'usager reste éligible. Sinon, il passe suspendu et la requête est transmise au module de gestion des disques.

Remarque : Si un article d'un fichier partagé a déjà été demandé par un usager A, la demande du même article par un usager B est annulée. Si la TPD de la B est alors vide, B reste éligible. Supposons qu'il soit élu avant A, dans ce cas, B sera à nouveau dérouté et redemandera le même article. B effectuera les transitions élu → éligible → élu jusqu'à ce que l'article arrive en mémoire physique et que la récupération soit possible.

Lancement et exécution de la requête

Ce sont les deux fonctions du module de gestion des disques :

- le lancement consiste à transformer la requête en commandes pour le disque RAD et à les ajouter aux commandes en attente provenant d'autres usagers.
- l'exécution de la requête consiste à prendre toutes ces commandes en attente, à les réarranger de manière à minimiser le temps de transfert sur RAD et à lancer le transfert (opération SIO) sur la chaîne de

commande ainsi obtenue. Par ailleurs, il faut pouvoir réactiver les usagers suspendus au fur et à mesure que leurs requêtes sont satisfaites. Pour cela, on note l'ordre dans lequel les usagers sont servis, une fois le réarrangement optimal fait.

L'exécution de la requête est lancée à chaque interruption de fin de transfert indiquant la fin d'une chaine de commande pour le disque. Toutefois, si aucun transfert n'est en cours à la fin du lancement de la requête, l'exécution en est demandée tout de suite.

Acquittement de la requête

Lorsque toutes les pages de la requête ont été transférées (vidage de l'ancien contenu si celui-ci a été modifié, puis lecture d'un nouvel article), l'usager peut à nouveau être candidat pour l'unité centrale. Il passe éligible.

Parallèlement, il faut activer tous les processus en attente de page. On l'obtient par l'opération D sur le sémaphore S:REQ de l'usager. Elle a pour effet de transférer tous les processus en attente de page dans la file des processus prêts de l'usager.

Nota : Vidage préventif de pages

Si après l'acquittement d'une requête, le disque est oisif, le système peut décider de recopier en mémoire secondaire un certain nombre de pages écrites qui ne sont plus utilisées.

Ce nombre doit être assez grand pour permettre une optimisation convenable des transferts, mais assez petit pour ne pas trop retarder l'exécution d'une requête pour le compte d'un usager suspendu. On pourra prendre par exemple un nombre de pages voisin du nombre moyen des pages d'une requête.

1.2.1.3 Etat Eligible

Un usager demeure éligible jusqu'à ce qu'il soit élu et qu'il reçoive l'unité centrale. Un usager est élu pour un intervalle de temps appelé quantum. Si l'usager n'est pas passé dans l'état suspendu en fin de quantum, l'unité centrale lui est otée et il repasse éligible.

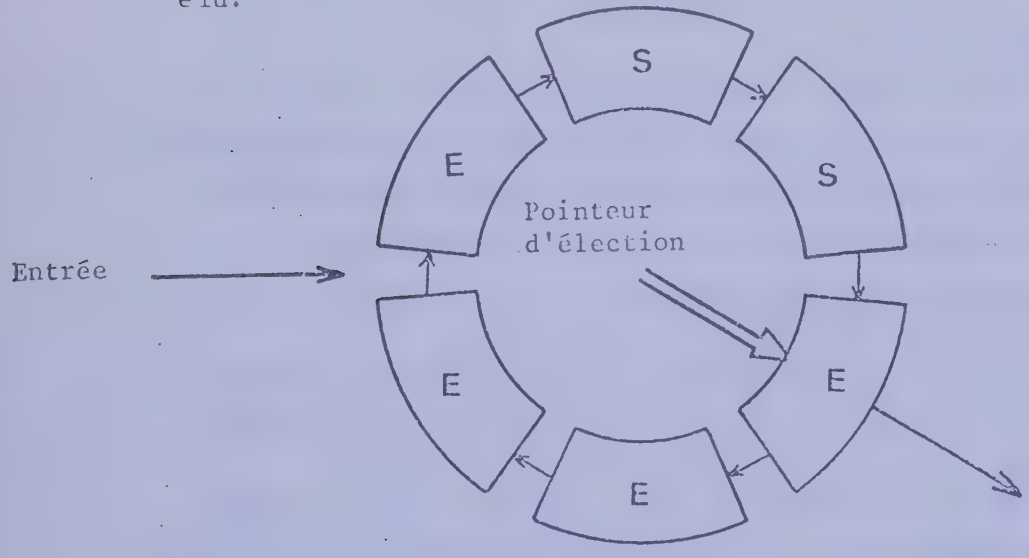
L'unité centrale est donnée successivement aux différents usagers éligibles.

1.2.1.4 Allocation de l'unité centrale

Les processus en compétition pour l'unité centrale sont :

- les processus prêts des usagers éligibles
- les processus prêts du moniteur. De même que pour chaque usager, il existe pour le moniteur une file de ses processus prêts.

Les usagers courants sont rangés dans une liste circulaire, la file C. A chaque usager de cette file est associé un mot d'état indiquant si l'usager est éligible ou suspendu. A tout instant, on distingué parmi les usagers éligibles un usager particulier l'usager élu.

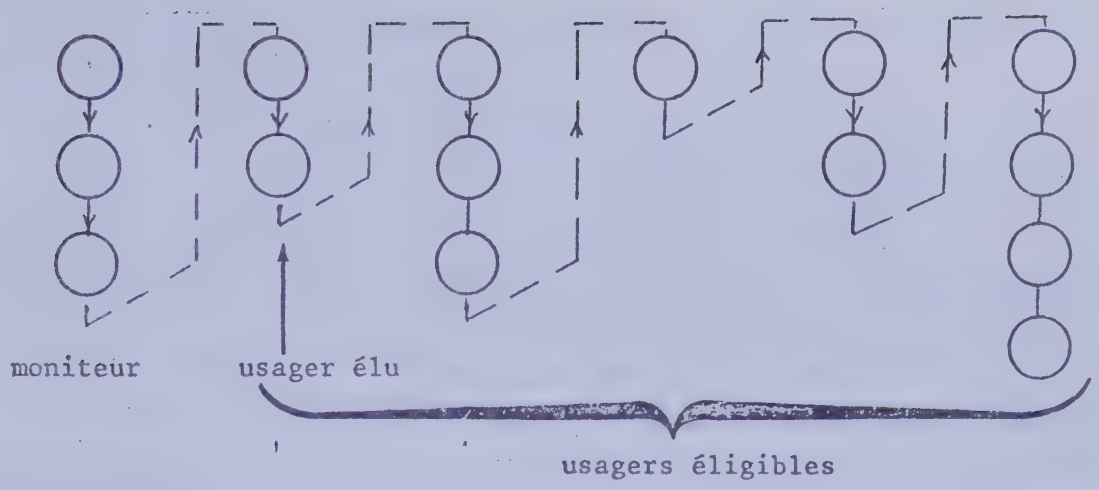


La file des usagers courants.
E : éligible
S : suspendu

Définissons alors une relation d'ordre entre les processus candidats à l'unité centrale :

- les usagers sont rangés dans l'ordre suivant : moniteur, usager élu, puis usagers éligibles dans l'ordre de la file c.
- les processus prêts du moniteur et des usagers sont rangés dans l'ordre de leur priorité, à l'intérieur de chaque usager (et du moniteur).

L'ensemble des processus condidats à l'unité centrale, ordonné par les relations définies ci-dessus est appelé la chaine d'activation.



On décide d'allouer à chaque instant l'unité centrale au premier processus de la chaine d'activation. Ceci revient à appliquer l'algorithme suivant :

on alloue l'unité centrale :

- au premier processus de la file prêt du moniteur, si celle-ci n'est pas vide,
- au premier processus de la file prêt de l'usager élu, sinon.

On a vu que cette dernière n'est jamais vide, car elle contient au moins le processus final. Quand celui-ci est activé, il appelle un processus du moniteur qui change d'usager élu en prenant le prochain éligible de la file C. L'algorithme d'allocation suit donc bien la chaine d'activation.

On doit changer l'allocation de l'unité centrale chaque fois que la chaine d'allocation varie c'est-à-dire :

- à la fin de chaque primitive P, V ou D de blocage et d'activation de processus
- à la fin des primitives de création ou de destruction de processus
- à chaque quantum, lorsque l'usager élu change
- lorsque l'usager élu quitte l'état élu pour passer suspendu, oisif ou prêt
- lorsqu'un nouvel usager devient éligible.

Comme les trois derniers cas se traduisent par des opérations V ou D sur des sémaphores du moniteur, on se ramène toujours à l'action d'une primitive. On inclue donc dans ces primitives, l'algorithme d'allocation que nous venons d'indiquer.

1.2.2 Boucle externe de circulation des usagers

Cette boucle contient trois états : courant, oisif et prêt.

1.2.2.1 Etat prêt, Réservation

Nous avons vu qu'un usager dans l'état prêt n'a encore aucune ressource et est demandeur de mémoire. Tous les usagers en attente de mémoire sont rangés dans une file appelée file P.

Pour faire passer un usager de l'état prêt à l'état courant (déplacement de la file P vers la file C), on essaie de lui réserver de la mémoire sur la base des catégories. A un instant donné, on associe à tout usager une catégorie qui est l'ensemble (NMAX, TMAX) d'un contingent maximal de pages physiques et d'un temps maximal de séjour dans la file des courants. Le nombre des catégories différentes et les caractéristiques NMAX et TMAX de chacune d'elle sont des paramètres du système contenus dans une table. Le choix des éléments de cette table se fait encore de manière empirique et est lié à l'idée qu'on se fait du comportement des usagers. Par expérimentation, on peut essayer de trouver une suite de valeurs satisfaisantes

La condition d'admission dans la file C est

$$NMAX \leq NDISP$$

où NDISP désigne le nombre de pages physiques disponibles, c'est-à-dire non réservées. Si cette condition est satisfaite l'usager entre dans la file C, où il reste au plus jusqu'à l'écoulement de son temps de résidence. Bien entendu NDISP est mis à jour.

La gestion de la file P se fait comme suit :

- on essaie d'admettre le premier usager de la file. S'il passe courant, le test d'admission est répété pour le nouvel usager en tête de file.
- si le test échoue pour le premier usager de la file, aucun usager n'est admis dans la file C. On n'essaie pas de trouver dans la file P, un usager ayant un NMAX suffisamment petit. On évite ainsi de faire attendre indéfiniment des usagers ayant un grand NMAX.

Le passage de prêt à courant peut être tenté quand il y a assez de pages disponibles, c'est-à-dire

- à l'arrivée d'un usager dans la file P alors que celle-ci est vide
- à la libération de mémoire physique consécutive au départ d'un usager de la file C.

1.2.2.2 Etat courant

Lorsqu'un usager passe de l'état prêt à l'état courant, il acquiert le droit de faire charger des pages dans la limite de sa catégorie. Une des décisions de base d'ESOPE (4e principe) est de déclencher ce chargement à la demande. Pour cela, l'usager doit recevoir l'unité centrale de manière à aller jusqu'à un défaut de page, seule façon de connaitre la demande. On fait donc entrer un usager dans la file courant à l'état éligible.

Préchargement

Quand un usager passe de prêt à courant, on dispose d'une certaine information sur son comportement futur. Il utilisera certainement les pages virtuelles

- dont le numéro figure dans les compteurs ordinaux des processus prêts de l'usager,
- qui ont provoqué un défaut de page mais que l'on n'avait pu charger, lors du précédent passage de l'usager en mémoire centrale.

Il utilisera très probablement les pages virtuelles de son "working set". On peut avoir une approximation de celui-ci par les n dernières pages référencées ou plus grossièrement, par les n dernières pages chargées (n est un paramètre expérimental).

Le préchargement consiste à utiliser cette information et à charger par anticipation les pages virtuelles, qui seront certainement utilisées. Le préchargement peut être étendu au cas des pages d'emploi probable. Cette extension est discutable dans la mesure où elle risque de charger les disques, qui sont déjà une ressource critique du système.

Si le préchargement est utilisé, un usager entre dans la file courant à l'état suspendu.

Dans la première version de l'allocateur, seul est emplanté le chargement des pages à la demande, mais lorsqu'un usager passe de courant à prêt, on garde trace de sa requête. On l'ajoutera à la première requête qu'il fera quand il sera redevenu courant. Dans les versions ultérieures, on ajoutera le préchargement des pages d'emploi certain, puis si cette modification se révèle positive, le chargement des pages d'utilisation probable.

Comptage du temps d'unité centrale utilisé

Un usager ne peut rester courant plus longtemps que le TMAX de sa catégorie. Pour s'en assurer, on utilise l'horloge du système qui fournit l'heure. A l'entrée dans l'état courant, on calcule pour

l'usager une date d'éjection (c'est : heure actuelle + TMAX). Cette date d'éjection est testée à chaque départ de l'état élu, soit par fin de quantum, soit par passage à l'état suspendu.

Si elle est atteinte ou dépassée l'usager passe à l'état prêt.

Comptage des pages utilisées

Un usager doit quitter l'état courant lorsqu'il a utilisé toutes les pages qui lui ont été allouées. On doit donc compter les pages qui lui sont allouées. Ceci est fait :

- dès le déroutement, en cas de récupération de page en mémoire
- lors de la préparation de la requête. On reporte le comptage à cet endroit pour éviter de compter deux fois les pages couplées au même article.

Une difficulté apparait pour compter les pages d'un fichier partagé entre plusieurs usagers. Si on ignore ce partage, les articles partagés sont comptés autant de fois qu'il y a d'usagers les partageant. La mémoire va être sous utilisée d'autant. D'autre part, si on ne comptabilise pas les pages des fichiers partagés, chaque usager pourra demander, et obtenir, le chargement d'un nombre d'articles supérieur à la taille de sa catégorie. Il est possible d'arriver dans une situation où tous les usagers courants demandent le chargement d'articles supplémentaires, et où il ne reste aucune page allouable (NDISP=0). Il y a blocage du système. Ce blocage est définitif, car dans ce cas le test de dépassement du temps de résidence ne se fait plus puisqu'il n'y a plus d'usager élu.

On retient la méthode suivante :

- le jeu des catégories, la réservation de pages pour un usager et le comptage des pages utilisées portent seulement sur les pages privées des usagers,
- la réservation de pages pour un fichier partagé est faite à part et n'est faite qu'une fois.

Un fichier partagé est caractérisé par le nombre fixe NF de ses pages. La réservation de mémoire pour un tel fichier se fait dans deux cas :

a) à l'entrée dans la file C d'un usager ayant ouvert ce fichier, alors qu'aucun usager courant ne l'a ouvert. On essaie d'admettre ensemble l'usager et le fichier. Le test d'admission est dans ce cas :

NMAX + NF ⩽ NDISP

Si le test réussit, on fait :

NDISP := NDISP - (NMAX + NF)

La réservation des NF pages demeure tant que le fichier reste ouvert pour un usager courant au moins. La notion de temps de résidence n'existe pas pour un fichier partagé.

b) lors de l'ouverture par l'usager élu d'un fichier partageable qui n'est pas déjà ouvert par un autre usager courant. On vérifie alors si

NF ⩽ NDISP

Si oui, NF pages sont réservées pour le fichier et on les retranche de NDISP. Sinon, l'usager demandeur est renvoyé dans la file P. On sera ramené au cas a) lors de sa réadmission dans la file C, sauf si entre temps la réser&ation de pages par le fichier a été déclenchée par un autre usager.

Nota : Fermeture d'un fichier privé

Comme les articles d'un fichier fermé sont automatiquement découplés et qu'ils ne peuvent plus être accédés, nous avons décidé de soustraire du compte des pages le nombre de pages de ce fichier qui avaient été allouées. En conséquence, les pages d'un fichier fermé sont libérées et rendues disponibles pour d'autres usagers.

Retour des pages réservées

Les pages réservées redeviennent disponibles

- si l'usager qui les a réservées quitte la file C
- si le fichier partagé qui les a réservées n'est plus ouvert pour aucun usager courant

NDISP est remis à jour. Il vaut NDISP + NMAX dans le premier cas, NDISP + NF dans le second. Les pages qui avaient été allouées consécutivement à la réservation sont alors libérées et deviennent disponibles pour d'autres usagers.

Remarque : la réservation séparée pour les usagers et les fichiers partagés est destinée à favoriser l'accès à ces derniers (compilateur CPL1 par exemple). Elle présente toutefois un inconvénient si NF est grand, celui d'immobiliser par la réservation un grand nombre de pages, car la taille de fichier est fixée et ne varie pas comme la taille NMAX. On est conduit à éclater de gros fichiers partagés en plusieurs fichiers. Une autre méthode peut être employée - elle consiste à ne pas tenir compte à la réservation, des fichiers partagés. Seul intervient NMAX. On établit deux comptes pour les pages allouées à un usager, un compte des pages privées et un compte des

pages partagées. Par ailleurs, à chaque fichier partagé, on associe un compte de pages allouées.

A la réservation on a NDISP := NDISP - NMAX

Au départ de la file C NDISP := NDISP + compte des pages privées allouées. On libère les pages privées.

Lorsque le fichier partagé n'est plus ouvert pour aucun usager courant NDISP := NDISP + compte des pages allouées au fichier partagé.

Départ de la file C

Pour éviter des chargements inutiles de pages, un usager ne quitte l'état courant qu'au départ de l'état élu.

Un usager quitte l'état courant lorsque tous ses processus sont bloqués intrinsèquement (il devient oisif) ou lorsqu'il a atteint une limite de sa catégorie (il passe prêt).

On ajuste la catégorie en fonction du comportement immédiat de l'usager :

- s'il a atteint son temps de résidence et que le nombre des pages allouées est nettement inférieur à NMAX, on lui affecte une catégorie inférieure (où NMAX est plus petit ou bien TMAX plus grand),
- s'il a atteint la taille NMAX bien avant d'avoir épuisé son temps de résidence, on lui attribue une catégorie supérieure (où NMAX est plus grand ou bien TMAX plus petit)
- s'il passe oisif, on lui affecte une catégorie telle qu'il puisse passer rapidement courant dès qu'il y aura un message au téléimprimeur. (catégorie où NMAX est petit).

Les pages de mémoire centrale que l'usager occupe sont rendues disponibles pour d'autres usagers, mais on ne déclenche aucun vidage d'article en mémoire secondaire en accord avec la politique ne consistant à déplacer des articles que le plus tard possible.

1.2.2.3 Etat oisif

Un usager reste oisif tant que ses processus sont bloqués intrinsèquement. Dès que l'un d'eux est débloqué par l'exécution d'une primitive V ou D, il passe dans la file des processus prêts de son usager. Celui-ci peut passer prêt.

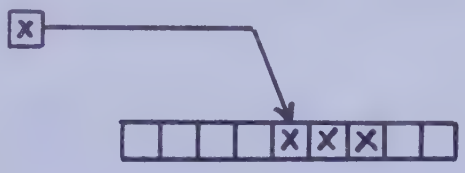
L'entrée dans la file P se fait selon le critère suivant :

- l'usager temps réel entre en tête
- l'usager pour le travail de fond vient en queue
- les autres usagers sont rangés par ordre d'entrée dans la file. Autrement

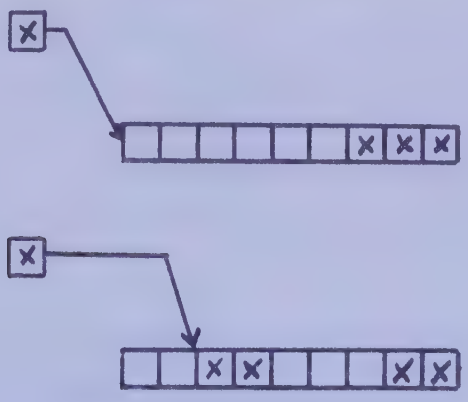
dit, l'entrée se fait en queue juste devant l'usager pour le travail de fond.

Regroupement

On peut essayer de favoriser le partage des fichiers en faisant passer successivement dans l'état courant tous les usagers partageant la même procédure. Pour cela, il suffit de regrouper ces usagers dans la file P. Quand un usager entre dans la file P, on y recherche la sous file des usagers qui partagent la même procédure que le nouveau venu. Si cette sous file existe, le nouveau venu est placé à sa queue, comme le montre la figure ci-dessous où les usagers partageant la même procédure sont marqués d'une croix.



Cependant, si le premier usager de la file P partage le fichier, les autres usagers marqués d'une croix peuvent se coaliser de manière à ce qu'aucun autre usager ne puisse passer courant. Il suffit que l'un d'eux soit toujours prêt. Pour éviter cela et garantir que tout usager prêt puisse passer dans l'état courant au bout d'un temps fini, le regroupement n'a lieu que si le premier usager de la sous file n'est pas en tête de la file P. D'où les deux cas de la figure suivante :



Cet algorithme est généralisable au cas de l'utilisation

simultanée par chaque usager de plusieurs procédures partageables, mais son implantation devient vite coûteuse.

De plus, une simulation a mis en évidence la faible longueur de la file P. L'amélioration apportée par l'algorithme de remplacement est alors négligeable . Aussi n'est-t-il pas implanté pour le moment dans ESOPE.

1 .3 Les processus de l'allocation

L'activité de l'allocateur a été décomposée en processus parallèles directement reliés aux transitions d'états des usagers. Tous ces processus sont résidents et permanents (ils ne sont jamais détruits). Ils s'exécutent en mode maître et en adresses physiques (sans topographie). Ils sont cycliques et commencent par une primitive P sur un sémaphore privé.

Leur état de repos est d'être bloqué derrière ce sémaphore.

1 .3.1 Processus FINAL

Soit X l'usager élu. D'après le principe de l'allocation multiple, tous ses processus prêts sont activés successivement. Ils sont bloqués soit intrinsèquement, soit technologiquement. Dans ce dernier cas, il y a déroutement pour défaut de page et ce déroutement engendre une primitive P(S:REQ(X) qui met le processus actif dérouté dans la file du sémaphore S:REQ(X) de l'usager X.

Pour signaler au moniteur que tous les processus de l'usager X sont désormais bloqués, nous avons introduit un processus FINAL qui est de priorité inférieure à celle des autres processus de l'usager et qui n'est jamais bloqué :

```
FINAL : begin
    A : désarmer l'interruption horloge du quantum ;
        V(S:PREP) ; comment  S:PREP est le sémaphore privé du processus
                             PREPARER-REQUETE qui est activé de suite parce
                             qu'il a une meilleure priorité (c'est un
                             processus du moniteur) ;
        goto A
        end
```

Le processus final et l'interruption d'horloge du quantum sont exclusifs l'un l'autre, car tous deux ont pour rôle d'appeler le moniteur

pour changer d'usager élu. C'est pourquoi d'une part final est masqué contre les interruptions, d'autre part il désarme l'interruption quantum qui n'a plus de raison d'être.

1.3.2 Processus PREPARER-REQUETE et interruption de quantum

Le rôle de PREPARER-REQUETE est de détecter l'état dans lequel doit passer l'usager élu. Pour cela, il doit préparer la requête de pages de l'usager (d'où son nom). Ce processus ne change que l'état d'un usager, il n'effectue pas les changements de file associés. C'est le processus CHANGEMENT qui les fera.

```
PREPARER-REQUETE : begin
            A : P(S:PREP) ;
                X := usager élu ;
                if E(S:REQ(X)) ≥ 0  then  état (X) := oisif
                else
                   begin
                   préparation de la requête de pages de l'usager ;
                   comment : la requête peut être vide ;
                   if requête vide then
                       begin
                       état (X) := éligible ;
                       D(S:REQ(X)) ; comment : toutes les pages demandées
                                     sont déjà en mémoire centrale
                       end
                   else
                       begin
                       if limite de catégorie atteinte (NMAX ou TMAX) then
                                                       état (X) := prêt
                       else
                           begin
                           état (X) := suspendu ;
                           lancement de la requête au disque ;
                           if disque oisif then V(S:EXEC) ; comment :
                              demande d'exécution de la requête
                           end
                       end
                    end ;
               V(S:CHANGE) ;
               goto A
               end
```

De même, l'interruption de quantum change l'état de l'usager puis appelle CHANGEMENT.

```
IT- QUANTUM : begin
              X := usager élu ;
              if temps de résidence atteint then état (X) := prêt
              else état (X):=éligible ;
              V(S:CHANGE)
              end
```

1.3.3 Processus CHANGEMENT

Ce processus effectue les changements de file préparés dans PREPARER-REQUETE et dans l'IT-QUANTUM. Certains d'entre eux entrainent la libération de mémoire. Dans ce cas, on regarde si en conséquence des usagers prêts peuvent devenir courants.

D'autre part CHANGEMENT est appelé très fréquemment ; il s'écoule au maximum un quantum entre deux activations de ce processus. On peut donc y effectuer un certain nombre de tests sur l'état du système. En particulier, on y regarde si des usagers marqués oisifs ont des processus prêts en plus de FINAL. Si oui, on fait passer ces usagers à l'état prêt. L'exécution de ce test de manière périodique dans CHANGEMENT évite de le faire systématiquement dans les primitives V et D. Le retard introduit est faible puisque le quantum d'unité centrale est petit (de l'ordre de 100 ms).

De même, les pages libérées à la fermeture d'un fichier ne sont pas utilisées tout de suite pour essayer d'entrer de nouveaux usagers courants. On attend CHANGEMENT qui le fait systématiquement.

On doit s'assurer que CHANGEMENT est toujours appelé, même dans le cas où il n'y a pas d'usager élu. Ceci peut être réalisé :
- en armant l'IT-QUANTUM même quand il n'y a plus d'usager élu
- ou en réalisant la boucle d'attente du système dans changement.

C'est la seconde solution que nous avons adoptée.

```
CHANGEMENT : begin
         A : P(S:CHANGE) ;
             X := usager élu ;
             if ETAT (X) = oisif then
                  begin
                  libérer les pages allouées à X ;
                  supprimer la réservation de pages pour X;
                  mettre X dans la queue des usagers oisifs
```

```
                end ;
        else
                if ETAT (X) = prêt then
                    begin
                    libérer les pages allouées à X ;
                    supprimer la réservation de pages pour X ;
                    mettre X dans la queue des usagers prêts
                    end ;

BOUCLE : comment : boucle d'attente du système lorsque l'unité centrale est
                   oisive. TOTAL est un nombre grand (de l'ordre de 250000) ;
        for N := 0 step 1 until TOTAL do
            begin
            introduction dans la file P des usagers oisifs dont un processus
            a été débloqué ;
            introduction dans la file C à l'état éligible d'autant
            d'usagers prêts que possible ;
            réservation de place au fur et à mesure pour ces usagers ;
            comment : recherche d'un usager à élire. SUC(Y) est l'usager
            suivant Y dans la file courant. SUCUSELU est l'usager de la
            file C qui suivait l'usager élu au précédent appel de change-
            ment. Cette variable intermédiaire est nécessaire à cause des
            changements de file : l'usager élu peut ne plus être courant ;
            Y := SUCUSELU ;
            repeat
                if état (Y) ≠ éligible then Y := SUC (Y)
                else
                    begin
                    usager élu := Y
                    SUCUSELU := SUC(Y) ; comment : préparation du prochain
                    appel de changement ;
                    charger la mémoire topographique et les protections
                    d'accès de l'usager Y ;
                    relancer l'horloge pour le quantum d'unité centrale ;
                    goto A
                    end
            until Y = SUCUSELU
            end ;
        message (ESOPE EN OMPHALOSKEPSIE) ;
        comment : omphaloskepsie : contemplation du nombril ;
```

```
goto BOUCLE
end
```

Notons que nous devons employer une variable intermédiaire SUCUSELU pour pouvoir entrer dans le file courant, même quand l'usager élu a été oté. Cette variable est entretenue par CHANGEMENT et elle suffit car le seul usager qui peut être oté de la file courant est l'usager élu (il est oté dans CHANGEMENT, et aussi au départ du système).

Cette file est circulaire et contient la tête de file. Lorsque l'usager élu est seul dans la file, SUCUSELU pointe vers la tête de file. On n'a pas à introduire de cas particulier pour cette situation. Toutefois, on doit vérifier qu'un élément utilisé n'est pas la tête de file.

1.3.4 Processus EXECUTER-REQUETE

C'est le processus de gestion du transfert avec les disques RAD. Il est appelé à chaque fin de transfert d'une chaine de commande ; ou dans PREPARER-REQUETE quand le disque est oisif.

```
EXECUTER-REQUETE :
                begin
            A : P(S:EXEC) ;
                prendre toutes les commandes en attente provenant des derniers
                lancements non satisfaits, les réarranger et les chaîner
                de manière à minimiser le temps de transfert sur RAD ;
                if pas de commande en attente then
                     if pages disponibles écrites then vidage préventif
                    else  disque oisif := true
                else lancement du transfert ;
                goto A
                end
```

1.3.5 Processus ACQUITTER-PEQUETE

On a vu que les commandes de transfert avec le disque sont chaînées. La dernière commande de chaque usager dans la chaîne est codée de manière à générer une interruption de fin de requête. Cette interruption appelle le processus ACQUITTER-REQUETE qui fait passer l'usager à nouveau éligible.

```
ACQUITTER-REQUETE : begin
                A : P(S:ACQ) ;
                    comment : X est l'usager dont la requête a été satisfaite ;
                    état (X) := éligible ;
                    mise à jour de la TPV (table des pages virtuelles) de
                    l'usager X ;
                    D(S:REQ(X) ; comment active tous les processus de X
                    bloqués technologiquement ;
                    goto A
                    end
```

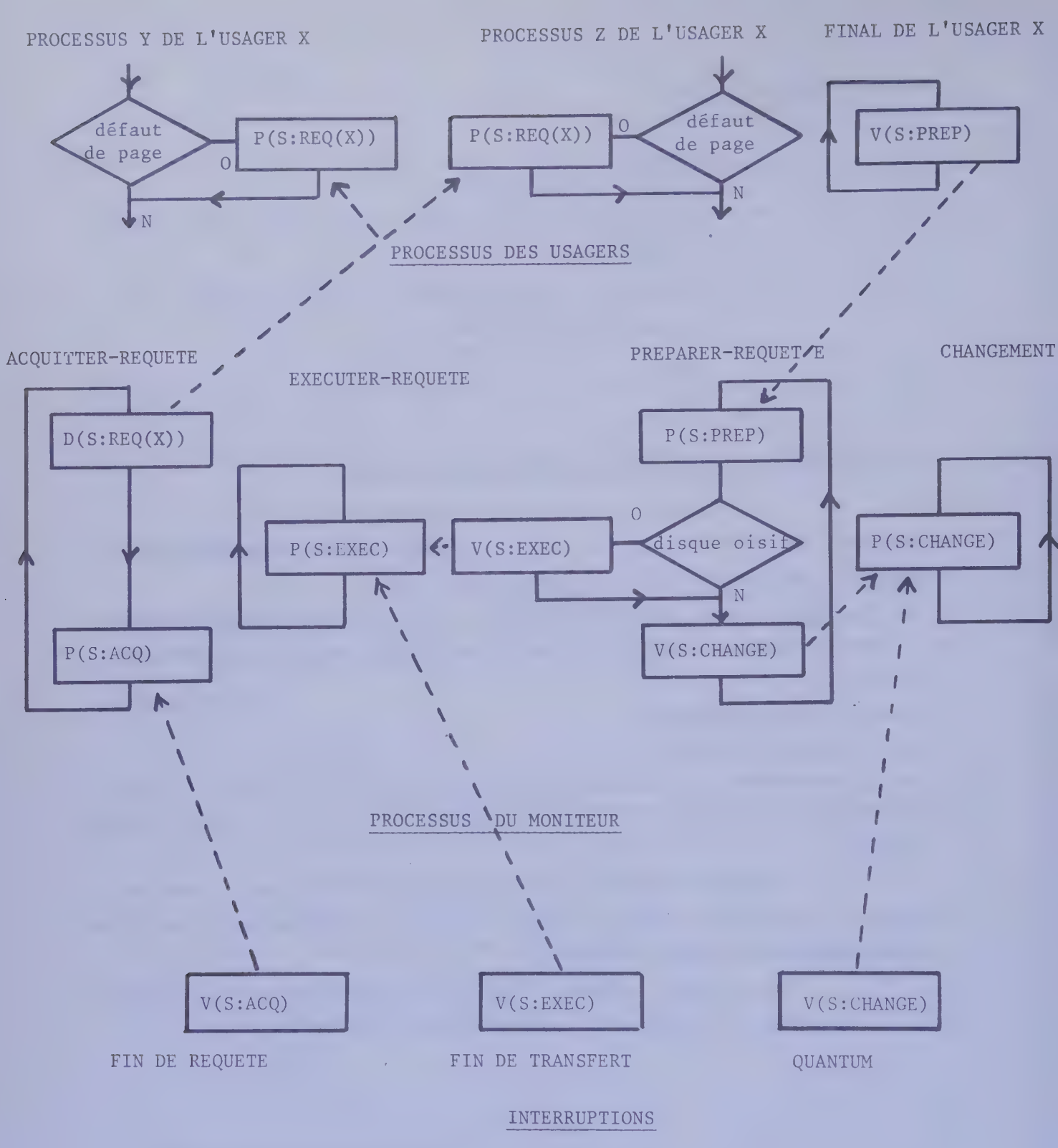
1.3.6 Coopération entre les processus de l'allocateur

La coopération entre ces processus est régie par les sémaphores. Nous avons aussi la possibilité d'attribuer une priorité aux processus. Nous avons choisi l'ordre suivant :

- le plus prioritaire, ACQUITTER-REQUETE. Il est en effet souhaitable de libérer le module disque des requêtes acquittées pour qu'il puisse en relancer,
- ensuite EXECUTER-REQUETE
- puis PREPARER-REQUETE
- enfin CHANGEMENT. Ce processus doit être le moins prioritaire car c'est lui qui contient la boucle d'attente du système lorsqu'il n'y a plus d'usager éligible.

Par ailleurs ces processus accèdent à des tables communes, un sémaphore d'exclusion mutuelle S:MUTEX1 est utilisé pour garantir qu'un seul de ces processus y accède à la fois.

La figure ci-dessous résume la coopération entre les processus. Le sémaphore S:MUTEX1 n'y figure pas, pour plus de clarté.



COOPERATION ENTRE LES PROCESSUS DE L'ALLOCATEUR

Les lignes pointillées indiquent les voies d'activation.

1.4 Mécanismes et tables de données

Nous étudierons essentiellement dans ce chapitre, les problèmes de gestion des pages physiques. Pour cela après avoir défini des états pour les pages et introduit une structure de données, nous décrivons l'évolution de ces états dans les divers mécanismes de l'allocateur .

1.4.1 Etat des pages physiques

Les mécanismes de gestion des pages physiques ont été clarifiés en introduisant la notion d'état d'une page.

Les pages résidentes sont attribuées en permanence au moniteur, elles ne sont l'objet d'aucune gestion. Les pages semi-résidentes sont attribuées à des usagers pour toute la durée de leur existence dans le système. Ce sont par exemple :

- des pages attribuées à l'usager "console graphique" utilisées comme mémoire de rafraichissement pour l'image.
- une page par usager conversationnel, contenant un tampon d'échange avec le téléimprimeur et diverses variables de communication entre l'usager et le moniteur.

Parmi les pages allouées à des fichiers d'usagers, on distingue les pages intactes qui n'ont fait l'objet d'aucune écriture depuis leur allocation, des pages modifiées. Les premières ont une image conforme sur disque RAD ; les secondes n'ont pas d'image conforme.

Pour chaque fichier interne, on entretient deux listes de pages :

- la liste de ses pages intactes,
- la liste de ses pages modifiées.

Une page allouée passe dans l'état non allouée dans deux cas :

- quand le fichier auquel elle appartient n'est plus ouvert par aucun usager
- quand aucun usager courant ne l'utilise plus.

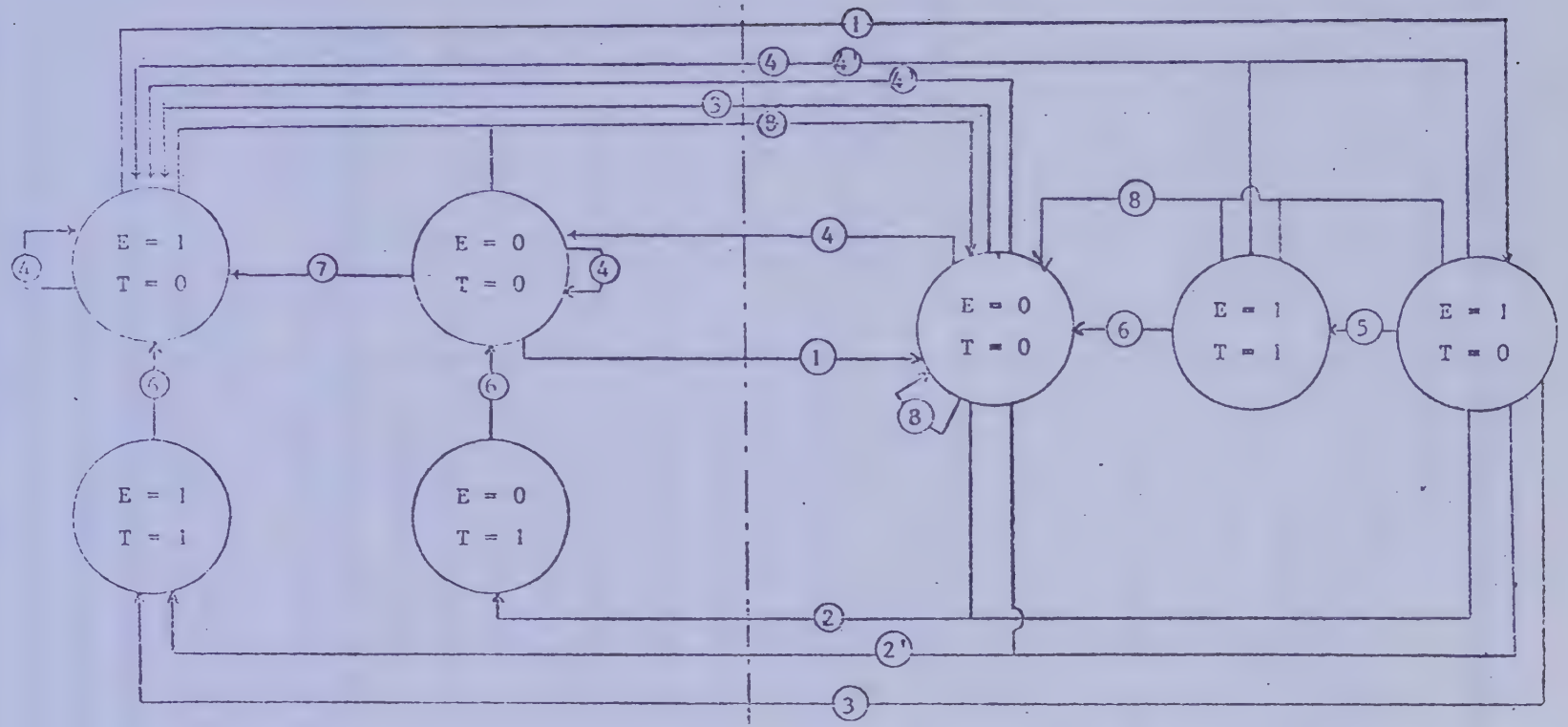
On distingue également parmi les pages non allouées, les pages libres des pages quasi-libres. Une page libre peut être

La figure suivante donne l'évolution d'une page physique entre les états introduits.

Nota : Comme les articles des fichiers partagés ne sont accessibles qu'en lecture ou en exécution, le diagramme de transition entre états de pages allouées à un fichier partagé est plus simple et est un sous ensemble du schéma général (seuls demeurent les états pour lesquels E = 0).

PAGES ALLOUEES (A = 1)

PAGES NON ALLOUEES (A = 0)



Légende :

1 - Libération
2 - Allocation avec chargement (accès lect/exec)
2'- Allocation (accès écriture) avec chargement
3 - Allocation pour amorçage (accès écriture) (pas de chargement)
4 - Récupération
4'- Récupération, accès écriture
5 - Vidage préventif
6 - Acquittement de requête
7 - Ecriture dans une page inexistante
8 - Destruction d'un fichier

... TOUS POSSIBLES ENTRE LES ETATS DES PAGES VIRTUELLES

Les numéros des transitions renvoient au tableau ci-dessous, où sont résumées leurs causes et les actions correspondantes.

Transition n°	Action	Circonstances
1	libération	fermeture de fichier départ d'un usager de l'état courant
2	allocation avec chargement	demande de page (1)
2'	idem avec accès en écriture	(1)
3	Allocation pour un nouvel article	accès (écriture permise) à un article inexistant
4	Récupération d'une page non allouée	demande de page
4'	idem accès écriture	
5	vidage préventif	disque oisif
6	arrivée d'un article	fin de transfert
7	changement de liste (intacte→modifiée)	première écriture dans un article déjà chargé (2)
8	libération avec perte du contenu	destruction de fichier

Remarques :

(1) des pages allouées non en transit peuvent être récupérées car plusieurs pages virtuelles peuvent être couplées au même article ; seule la première référence donne lieu à un chargement.

(2) lorsque le premier accès à un article non chargé est une écriture, la page correspondante est directement placée dans la liste des pages modifiées. La transition 7 ne concerne donc que des pages déjà chargées pour lecture ou exécution.

I .4.2 Structures de données pour la gestion des pages physiques

Tables des pages physiques (TPP).

L'état dans lequel se trouve toute page physique est, à tout instant, enregistré dans cette table. Pour avoir un point de référence, les informations contenues dans cette table sont toujours exactes, c'est à dire qu'elles sont mises à jour dès qu'une page subit un changement d'état.

Chaque élément de la TPP, repéré par un numéro de page physique contient :

- un champ FTPP (numéro de fichier)
- un champ ATPP (numéro d'article)
- les bits d'allocation A, E, T
- des liens de chainage SUC et PRED

Table des pages virtuelles (TPV) (cf aussi chapitre 3.4.2 de la 2e partie)

Chaque élément est repéré par le numéro de page virtuelle. Il y a une TPV par usager.

- FTPV : numéro du fichier
- ATPV : numéro d'article du fichier couplé à la page virtuelle
- PE : protection effective de la page
- PF : protection d'accès à la page (demandée par l'utilisateur)
- PTPV : numéro de page physique support de la page virtuelle.

Table des fichiers opérationnels (TFO) (cf aussi chapitre 3.4.2 de la 2e part)

Chaque élément correspond à un fichier interne F et comprend :

- NUSL : nombre d'usagers ayant ouvert le fichier
- NUSC : nombre d'usagers courants partageant le fichier
- NPI, QPI : nombre total de pages et pointeur de queue de la liste des pages intactes allouées au fichier
- NPE, QPE : nombre total de pages et pointeur de queue de la liste des pages écrites allouées au fichier
- PTTIF : pointeur sur la TIF du fichier

Table d'implantation du fichier (TIF)

Chaque entrée correspond à un article de fichier. Il y a une TIF par fichier interne.

- PTIF : numéro de page physique
- DTIF : emplacement original de l'article sur RAD
- U : bit d'utilisation. Il indique si l'article contient une information significative ou non.

On consigne dans la TIF les possibilités d'accès à un article. En conséquence elle est tenue à jour comme si les transferts étaient instantanés.

Table des pages demandées (TPD)

Chaque entrée correspond à une demande. Il y a une TPD par usager.

- PTPF : numéro de page physique
- VTPD : numéro de page virtuelle
- FTPD, ATPD : coordonnées d'un article
- PFTPD : protection d'accès à la page
- PETPD : protection effective à mettre dans la TPV à l'acquittement de la requête
- R : type d'accès à l'article (R=1 pour écriture)
- VP : bit indiquant si la page allouée à VTPD doit être vidée au préalable
- NA : bit d'amorçage (première écriture dans l'article de fichier)

Variables globales

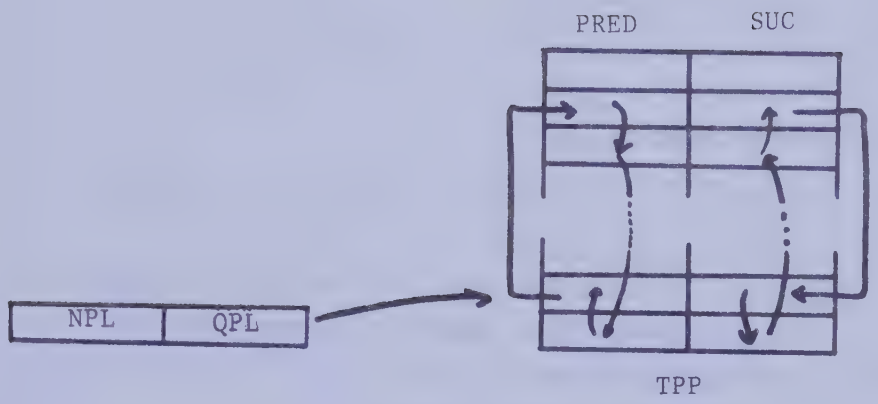
NPL, QPL : nombre total de pages et pointeur de queue de la liste des pages libres

NPQL, QPQL : nombre total des pages et pointeur de queue de la liste des pages quasi-libres

NPV, QPV : nombre total de pages et pointeur de queue de la liste des pages en vidage préventif.

Ces pointeurs et ceux de la TFO repèrent des listes de pages physiques chainées par les liens SUC et PRED.

Par exemple :



Remarquons :

- quand une liste est vide, le pointeur de queue à la valeur 0 (la page physique 0, réservée au moniteur, ne fait partie d'aucune liste)

- quand une page physique X n'appartient à aucune liste (cas des pages résidentes et semi-résidentes), on a SUC(X) = PRED(X) = 0.

1.4.3 Déroutement pour violation de protection virtuelle

Nous ne considérons ici que l'utilisation de la protection virtuelle pour la gestion des pages.

1.4.3.1 Premier accès à une page virtuelle

La protection effective PE de cette page est à 3, ce qui a causé le déroutement. L'analyse du déroutement permet de trouver le numéro de page virtuelle V et de savoir si l'accès se fait en

écriture ou non. Si la protection du fichier le permet (PF = 0 en cas d'écriture, PF ≠ 3 sinon) on note dans la TPD à l'entrée courante X :

```
VTPD(X) := V ;
FTPD(X) := FTPV(V) ;
ATPD(X) := ATPV(V) ;
PFTPD(X) := PFTPV(V) ;
R(X) := if écriture then 1 else 0 ;
NA(X) := U(FTPV(V), ATPV(V))
```

Nota : pour simplifier l'exposé, nous ne considérons pas le cas où il y aurait récupération.

1.4.3.2 Ecriture dans une page encore intacte

La protection effective PE est à 1, d'où le déroutement. La protection du fichier PF doit être à zéro. Dans ce cas, la page est transférée, dans la liste des pages modifiées de son fichier. Comme plusieurs pages virtuelle peuvent correspondre à la même page physique, il est possible que la page soit déjà dans cette liste par suite d'une écriture dans une autre page virtuelle.

D'où l'algorithme :

```
P:= PTPV(V) ; F:= FTPV(V) ;
if E(P)= 0 then
     begin
     transférer P de QPI(F) dans QPE(F) ;
     NPI(F) := NPI(F) - 1 ;
     NPE(F) := NPE(F) + 1 ;
     E(P) := 1
     end ;
PE(V) := 0
```

1.4.4. Récupération d'un article

Rappelons que la récupération est obligatoire dès qu'un article de fichier se trouve en mémoire centrale. La récupération peut avoir lieu dès le déroutement ou bien lors de la préparation de la requête. Dans les deux cas, on dispose des coordonnées de l'article de fichier couplé à la page V. En les utilisant on remonte au champ PTIF correspondant dans la TIF. Si PTIF n'est pas nul, l'article est soit

présent en mémoire soit en cours de chargement (on verra plus loin que PTIF est rempli dès qu'une page physique est allouée).

A l'aide de PTIF, on se reporte à la table des pages physiques où les valeurs des bits A, E, T permettent de déterminer l'état de la page et la liste dans laquelle elle se trouve :

a) A = 1, E = 1, T = 0 : liste des pages écrites du fichier. L'article du fichier est couplé à une autre page virtuelle pour laquelle le défaut de page s'est déjà produit. Elle est déjà récupérée.

b) A = 1, E = 0, T = 0 : liste des page intactes du fichier. Ici encore l'article est couplé à une autre page virtuelle. La récupération est déjà faite. Si de plus l'accès tenté est une écriture et qu'écrire dans la page virtuelle est permis, on fait passer la page dans la liste des pages écrites de son fichier.

c) A = 0, E = 1, T = 0 : liste des pages quasi-libres. La page passe dans la liste des pages écrites du fichier.

d) A = 0, E = 0, T = 0 : liste des pages libres. La page passe dans la liste des pages intactes du fichier ou, s'il s'agit d'une écriture, dans la liste des pages écrites du fichier.

e) A = 0, E = 1, T = 1 : liste des pages en vidage préventif. Dans ce cas la récupération n'est pas obligatoire, mais elle permet d'éviter un transfert de RAD à mémoire centrale. La page récupérée doit passer dans la liste des pages écrites de son fichier. En effet si on la faisait passer dans la liste des pages intactes, on pourrait avoir la suite d'événements suivants :

- demande de vidage préventif de PTIF vers DTIF
- récupération et passage dans la liste des pages intactes
- départ de l'usager de l'état courant, libération de la page PTIF et passage dans la liste des pages libres
- allocation de la page PTIF à un autre usager et demande de chargement d'un autre article depuis DTIF1
- optimisation des demandes par le processus EXECUTER-REQUETE. Cela a pour conséquence de faire

passer le transfert avec DTIF1 avant celui avec DTIF

- chargement de DTIF1 dans PTIF
- vidage de PTIF dans DTIF

On aboutirait à perdre le contenu initial de PTIF et à recopier DTIF1 dans DTIF !

f) A = 1, E = 1, T = 1 ou A = 1, E = 0, T = 1. C'est une page dont le chargement a été demandé. Nous devons considérer deux cas :

- le fichier est privé : le chargement ne peut avoir été demandé que par un autre processus du même usager. Comme le système fonctionne en activation multiple, que les demandes de chargement sont regroupées dans la TPD, on élimine ce cas en supprimant les demandes multiples dans la TPD au début de la préparation de la requète (cf. § 5.2.1.2)
- le fichier est partagé : le chargement a été demandé par un processus d'un autre usager. La récupération est impossible car l'article n'est pas en mémoire.

Une fois la page récupérée, on note dans la TPD (champ PETPD) la nouvelle valeur de PETPV.

PETPV := if PFTPV > 0 then PFTPV else if écriture then 0 else 1

L'organigramme de la récupération s'écrit alors :

```
begin
F := FTPD(X) ; comment : X entrée courante dans la TPD ;
P := PTIF(F, ATPD(X)) ;
Y := A(P) * 4 + E(P) * 2 + T(P) ; Y := Y + 1 ;
case Y of
     begin comment : cas d)
     A(P) := 1
     if (PFTPD(X) = 0) & (R(X) = 1) then
          begin
          transférer P de QPL vers QPE(F) ;
          NPL := NPL - 1 ;
          NPE(F) := NPE(F) + 1
          E(P) := 1
          end
```

```
        else
            begin
            transfert P de QPL vers QPI(F) ;
            NPL := NPL - 1 :
            NPI(F) := NPI(F) + 1
            end
        end
or
        begin comment : cas impossible  end
or
        begin comment : cas c) ;
        A(P) := 1 ;
        transférer P de QPQL vers QPE(F) ;
        NPQL := NPQL - 1 ;
        NPE(F) := NPE(F) + 1
        end
or
        begin comment : cas e) ;
        A(P) := 1 ;
        T(P) := 0 ;
        transférer P de QPV vers QPE(F) ;
        NPV := NPV - 1 ;
        NPE(F) := NPE(F) + 1
        end
or
        begin comment : cas b) ;
        if (PFTFD(X) = 0) & (R(X) = 1) then
            begin
            transférer P de QPI(F) vers QPE(F) ;
            NPI(F) := NPI(F) - 1 ;
            NPE(F) := NPE(F) + 1
            end
        end
or
        begin comment : cas f) ;
        PTPD(X) := 0 ; comment: indique que l'article n'est pas en mémoire
        PETPD(X) := 3 ; comment : ainsi le prochain accès à cette
                                  page causera un déroutement ;
```

```
                goto FIN
                end
        or
                begin comment cas a) end
        or
                begin comment cas f) avec écriture dans un fichier partagé ;
                PETPD(X) := 3 ;
                goto FIN
                end ;
        PETPD(X) := if PFTPD(X) > 0  then PFTPD(X) else if R(X) = 1
                                     then 0 else 1 ;
FIN :   end
```

1.4.5 Allocation de page physique

Nous appelons allocation de page l'opération qui consiste à délivrer une page physique à un fichier. Au cours de l'allocation, la page va changer de contenu. Nous appelons contenu ancien celui qu'elle a avant allocation et contenu nouveau celui qui remplace l'ancien.

Les pages à allouer sont prélevées :

- dans la liste des pages libres, tant qu'il y en a . Le contenu ancien d'une page n'a pas à être préservé. Le chargement éventuel de l'article est dit inconditionnel.

- puis dans la liste des pages quasi-libres. Le contenu ancien d'une page doit être sauvegardé avant toute nouvelle utilisation. Le chargement éventuel de l'article est dit conditionnel.

Dès qu'une page passe à l'état allouée, elle est immédiatement chaînée dans une des listes (intacte ou écrite) du fichier correspondant à son contenu nouveau. Son contenu ancien ne peut plus être récupéré.

1.4.5.1 Allocation d'une page libre

Cette allocation s'accompagne des opérations suivantes :

- rendre inaccessible le contenu ancien de la page. Si ce contenu ancien correspond à un article de fichier, le champ PTIF de l'article dans la TIF de ce fichier pointe vers cette page. On a vu que cela permet la récupération. On remet ce champ PTIF à zéro.

- rendre accessible le contenu nouveau de la page.

Pour cela on met à jour la TIF correspondant au nouveau fichier : PTIF prend une valeur égale au numéro de la page allouée

- mettre à jour la TPP. C'est à dire :
  - inscrire les coordonnées FTPP et ATPP du nouveau contenu
  - allouer la page (A := 1)
  - indiquer le transfert (T := 1) si un changement doit avoir lieu. Sinon, si c'est la première fois qu'on écrit dans l'article (création d'article, U = 0 dans la TIF), la place réservée sur disque DTIF n'a pas encore été utilisée et elle ne contient aucune information significative. Pour avoir un état de référence de l'article, on remet à zéro tous les mots de la page.
  - indiquer l'écriture (E := 1) si c'est le cas
  - faire passer la page dans l'une des listes (pages intactes ou pages écrites) du fichier

1.4.5.2 Allocation d'une page quasi-libre

Comme nous voulons maintenir des informations toujours exactes dans la TPP, les actions s'effectuent en trois temps :

- à l'allocation de la page
- à la fin du vidage du contenu ancien de la page
- à la fin du chargement du contenu nouveau.

On traitera plus loin des deux derniers, en analysant l'acquittement de la requête. Au moment de l'allocation, on se borne :

- à rendre inaccessible le contenu ancien de la page
- à rendre accessible le contenu nouveau de la page
- à mettre à jour le TPP c'est à dire :
  - allouer la page
  - indiquer le transfert
  - indiquer l 'écriture si elle a lieu
  - faire passer la page dans l'une des listes du fichier

Nota : Le chargment d'une page quasi-libre est conditionnel. Cela veut dire que l'ordre des demandes concernant une page p devra être respecté par le module de gestion des disques

- vidage sur disque du contenu de p
- chargement depuis le disque de la page p.

1.4.5.3 Allocation d'une page en vidage préventif

On pourrait penser qu'en l'absence de page libre, il est préférable d'allouer une page en vidage préventif plutôt qu'une page quasi-libre, car cela permettrait d'éviter de déclencher un nouveau vidage.

En fait le vidage préventif pose des problèmes délicats :

- si le chargement de l'article nouveau est conditionnel. Or la demande de vidage préventif ne figure pas dans la requête de l'usager (c'est une requête de moniteur), tandis que la demande de chargement conditionnel s'y trouve. Comme le module de gestion des disques respecte l'ordre des demandes concernant une même page, en cas de chargement conditionnel, le vidage sera bien effectué avant le chargement. Toutefois, pour maintenir des informations exactes dans la TPP, il va falloir mettre en oeuvre une procédure compliquée à la fin de chaque vidage préventif (il faut remonter à la TPD de l'usager auquel la page est allouée).

- si l'article nouveau est une création. On ne demande alors aucun changement. Supposons par ailleurs que cette demande soit la seule de l'usager. Celui-ci ne lance aucune requête. Avec le mécanisme normal, il redevient éligible. Or il faut en fait qu'il reste suspendu jusqu'à la fin du vidage préventif.

Pour éviter ces inconvénients, nous avons décidé de ne jamais allouer de page en vidage préventif.

Cette décision a une conséquence sur le vidage préventif. En effet la réservation de pages pour un usager se sert de NDISP, qui indique le nombre de pages non réservées pouvant être allouées. Or on vient de voir que les pages en vidage préventif ne peuvent pas être allouées. En conséquence, pour éviter de mettre l'allocation de page en échec (NPL = 0 et NPQL = 0 lors d'une demande de page), on doit ôter de NDISP les pages en vidage préventif. Ceci conduit à ne vider ainsi qu'un tout petit nombre de pages.

Donc avant de vider préventivement N pages :

```
if (NDISP - N) ⩾ 0 then
    begin
    lancer le vidage préventif de N pages
    NDISP := NDISP - N
    end
```

et en fin de vidage

```
NDISP := NDISP + N
```

1.4.5.4. Organigramme de l'allocation

```
begin
comment : choix d'une page à allouer ;
if NPL > 0 then
    begin
    P := première page de QPL ; comment : page libre ;
    NPL := NPL - 1 ;
    VP(X) := 0 ; comment : X entrée de la TPD pour laquelle est
                           faite la demande ;
    end
else
    begin
    if NPQL ⩽ 0 then arrêt du système pour erreur ;
    P := première page de QPQL ; comment : page quasi-libre ;
    NPQL := NPQL - 1 ;
    VP(X) := 1 ;
    D := DTIF(FTPP(P), ATPP(P)) ; comment adresse de vidage de
                                    la page ;
    end ;
PTPD(X) := P ;
comment : transfert de la page dans l'une des listes du fichier ;
F := FTPD(X) ;
if R(X) = 1 then
    begin
    mettre P dans QPE(F) ;
    NPE(F) := NPE(F) + 1
    end
else
    begin
    mettre P dans QPI(F) ;
    NPI(F) := NPI(F) + 1
    end
```

```
comment : rendre inaccessible l'ancien contenu, s'il a un sens ;
if FTPP(P) ≠ 0 then PTIF(FTPP(P), ATPP(P)) := 0 ;
comment : rendre accessible le nouveau contenu ;
PTIF(F,ATPD(X)) := P ;
comment : mise à jour de la TPP et demandes de transfert. On utilise
3 compteurs IN, OUT, NCC, initialement à zéro, comptabilisant
respectivement le nombre de chargements (conditionnels ou non),
de vidages et de chargements conditionnels :
A(P) := 1 ;
if VP(X) = 0 then
     begin  comment : page libre
     FTPP(P) := F ;
     ATPP(P) := ATPD(X) ;
     if NA(X) = 1 then
          begin comment : amorçage d'un article, pas de chargement,
                          pas de vidage ;
          E(P) := 1 ;
          T(P) := 0 ;
          écrire zéro dans tous les mots de la page P ;
          goto FIN
          end ;
     end
else
     begin comment: page quasi-libre ;
     faire une demande de vidage de la page P à l'adresse D du disque;
     OUT := OUT + 1 ;
     if NA(X) = 1 then
          begin comment : amorçage d'un article, pas de change-
                          ment, mais vidage ;
          E(P) := 1 ;
          T(P) := 1 ;
          goto FIN
          end ;
     else NCC := NCC + 1 ;
     end ;
```

```
        faire une demande de chargement de la page P par la page
        DTIF(F,ATPD(X)) ;
        IN := IN + 1 ;
        T(P) := 1 ;
        E(P) := if  PETPD(X) = 0  then 1 else  0 ;
FIN : end
```

1.4.6 Exécution du transfert des pages

1.4.6.1 Contraintes

Pour minimiser le temps de transfert entre RAD et unité centrale, on est amené à réarranger l'ordre des pages à transférer en fonction des adresses disques. Toutefois il existe un certain nombre de contraintes.

La première concerne les demandes conditionnelles. Deux transferts s'adressent à la même page physique et l'ordre suivant doit être respecté :

- vidage sur disque de la page P
- chargement de la page P

Le module de gestion du disque respecte cette contrainte en travaillant en deux étapes. Lors d'une première exécution de requête, il lance l'opération de vidage. Ce n'est que dans l'exécution suivante (SIO suivant) qu'il lance l'exécution du chargement. Aussi, quelle que soit l'optimisation faite, on est certain que l'ordre indiqué est respecté.

La seconde concerne les demandes destinées à la même adresse sur disque. Supposons qu'un usager U1 dont la page P1 contient l'article (A,F), passe à l'état bloqué. Cette page P1 rejoint alors la liste des pages quasi-libres et devient allouable à d'autres usagers. Soit U2 l'usager qui la reçoit pour y mettre l'article (B,G). Il envoie une requête au disque pour cela. Supposons que le module de disque soit alors en train de traiter des requêtes correspondant à la $(i-1)^e$ exécution. A la ième exécution, il fera le vidage de P1 dans (A,F) et à la $(i+1)^{è}$ le remplissage de P1 depuis (B,G).

Supposons maintenant que l'usager U1 redevienne courant, qu'il redemande l'article (A,F), que l'allocateur lui attribue une page libre P2 et qu'il demande le chargement de (A,F) dans P2.

Supposons que cela se soit passé avant la fin de la $(i-1)^{è}$ exécution du transfert des disques. La $i^{è}$ exécution fera le chargement de P2 depuis (A,F). On constate alors qu'à la $i^{è}$ exécution on aura, pour U1, le vidage de P1 dans (A,F) et pour U2, le chargement de P2 depuis (A,F). L'ordre des requêtes doit être impérativement respecté puisque la version à jour de l'article se trouve dans P1 avant la $i^{è}$ exécution et non dans (A,F).

Dans l'exemple ci dessus, on pourrait éviter les transferts sur le disque en faisant passer l'information de P1 vers P2 directement et en mettant P2 dans la liste des pages écrites du fichier F.

Nous ne l'avons pas fait dans la première version d'ESOPE, dans un souci de simplicité et pour découpler le module de gestion des disques de l'allocation des pages.

1.4.6.2 Caractéristiques des RAD

La structure des disques RAD de notre système se présente comme suit. Il existe 3 disques gérés par un controleur unique. A tout moment, il ne s'exécute qu'un seul transfert. Chaque disque comprend 512 pistes de 16 secteurs numérotés de 0 à 15. Le secteur 15 de la piste K et le secteur 0 de la piste K + 1 sont considérés comme contigus. Un secteur comprend 360 octets.

Dans notre système, le transfert se fait par blocs de 6 secteurs contigus, correspondant à 540 mots, soit un peu plus d'une page de mémoire. De tels blocs de 6 secteurs sont également appelés pages de disque.

Il en résulte de cette disposition que les débuts de page peuvent se trouver à 8 emplacements de secteurs possibles sur les 16 (voir figure N), ce qui permet de répartir les pages en 8 files (F0 à F7).

```
        faire une demande de chargement de la page P par la page
        DTIF(F,ATPD(X)) ;
        IN := IN + 1 ;
        T(P) := 1 ;
        E(P) := if  PETPD(X) = 0  then 1 else  0 ;
FIN : end
```

1.4.6 Exécution du transfert des pages

1.4.6.1 Contraintes

Pour minimiser le temps de transfert entre RAD et unité centrale, on est amené à réarranger l'ordre des pages à transférer en fonction des adresses disques. Toutefois il existe un certain nombre de contraintes.

La première concerne les demandes conditionnelles. Deux transferts s'adressent à la même page physique et l'ordre suivant doit être respecté :

- vidage sur disque de la page P
- chargement de la page P

Le module de gestion du disque respecte cette contrainte en travaillant en deux étapes. Lors d'une première exécution de requête, il lance l'opération de vidage. Ce n'est que dans l'exécution suivante (SIO suivant) qu'il lance l'exécution du chargement. Aussi, quelle que soit l'optimisation faite, on est certain que l'ordre indiqué est respecté.

La seconde concerne les demandes destinées à la même adresse sur disque. Supposons qu'un usager U1 dont la page P1 contient l'article (A,F), passe à l'état bloqué. Cette page P1 rejoint alors la liste des pages quasi-libres et devient allouable à d'autres usagers. Soit U2 l'usager qui la reçoit pour y mettre l'article (B,G). Il envoie une requête au disque pour cela. Supposons que le module de disque soit alors en train de traiter des requêtes correspondant à la $(i-1)^e$ exécution. A la iè exécution, il fera le vidage de P1 dans (A,F) et à la $(i + 1)^è$ le remplissage de P1 depuis (B,G).

Supposons maintenant que l'usager U1 redevienne courant, qu'il redemande l'article (A,F), que l'allocateur lui attribue une page libre P2 et qu'il demande le chargement de (A,F) dans P2.

Supposons que cela se soit passé avant la fin de la $(i-1)^{è}$ exécution du transfert des disques. La $i^{è}$ exécution fera le chargement de P2 depuis (A,F). On constate alors qu'à la $i^{è}$ exécution on aura, pour U1, le vidage de P1 dans (A,F) et pour U2, le chargement de P2 depuis (A,F). L'ordre des requêtes doit être impérativement respecté puisque la version à jour de l'article se trouve dans P1 avant la $i^{è}$ exécution et non dans (A,F).

Dans l'exemple ci dessus, on pourrait éviter les transferts sur le disque en faisant passer l'information de P1 vers P2 directement et en mettant P2 dans la liste des pages écrites du fichier F.

Nous ne l'avons pas fait dans la première version d'ESOPE, dans un souci de simplicité et pour découpler le module de gestion des disques de l'allocation des pages.

1.4.6.2 Caractéristiques des RAD

La structure des disques RAD de notre système se présente comme suit. Il existe 3 disques gérés par un controleur unique. A tout moment, il ne s'exécute qu'un seul transfert. Chaque disque comprend 512 pistes de 16 secteurs numérotés de 0 à 15. Le secteur 15 de la piste K et le secteur 0 de la piste K + 1 sont considérés comme contigus. Un secteur comprend 360 octets.

Dans notre système, le transfert se fait par blocs de 6 secteurs contigus, correspondant à 540 mots, soit un peu plus d'une page de mémoire. De tels blocs de 6 secteurs sont également appelés pages de disque.

Il en résulte de cette disposition que les débuts de page peuvent se trouver à 8 emplacements de secteurs possibles sur les 16 (voir figure N), ce qui permet de répartir les pages en 8 files (F0 à F7).

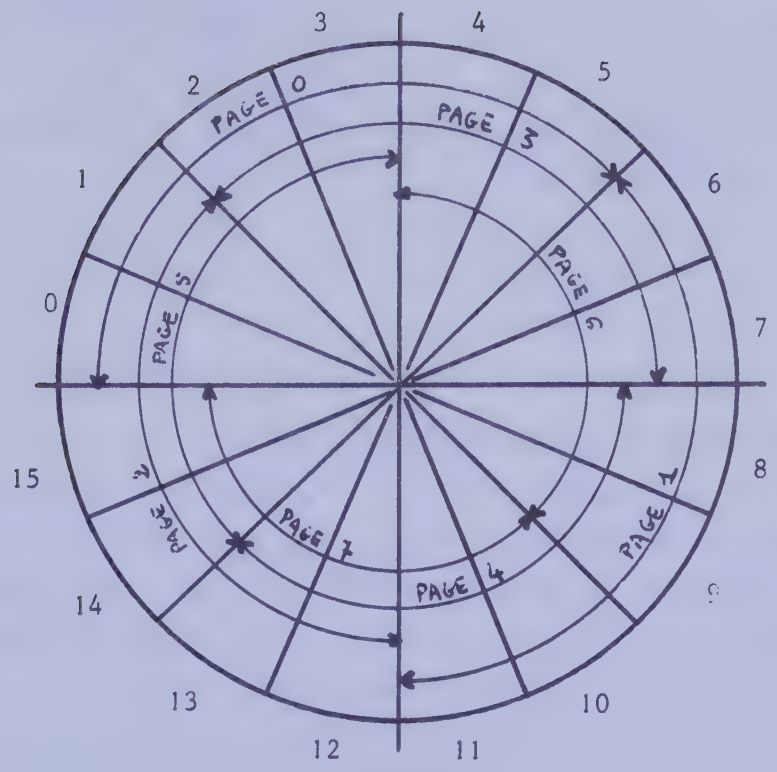


figure N - Pages et secteurs sur le RAD

file	secteurs	débuts de pages
F0	0	0, 8,...,8K,...
F1	2	3,11,...,8K+3,...
F2	4	6,14,...,8K+6,...
F3	6	1,9,....,8K+1,...
F4	8	4,12,...,8K+4,...
F5	10	7,15,...,8K+7,...
F6	12	2,10,...,8K+2,...
F7	14	5,13,...,8K+5,...

Une page (elle commence au secteur 2i) de numéro P appartient à la file Fi, où i est le reste de la division de 3P par 8.

A tout instant, il est possible de savoir, grâce à un ordre spécial, sur quel secteur se trouvent les têtes de lecture et écriture.

Les performances des disques sont les suivantes, sous alimentation à 50 Hertz :

- vitesse 1478 t/mn, soit un tour en 40,6 millisecondes
- intervalle de temps entre la fin d'un secteur et le début du suivant : 216 microsecondes
- temps de transfert : 15,6 millisecondes par page.

1.4.6.3 Optimisation du transfert sur un disque

Pour transférer le plus de pages possibles à chaque tour de disque, on est amené à ordonner les pages suivant la file (F0,F1,... F7) à laquelle elles appartiennent et à chaîner les commandes en prenant une page par file dans l'ordre cyclique suivant :

F0,F3,F6,F1,F4,F7,F2,F5,F0,F3,.....

Ce schéma peut recevoir deux améliorations :

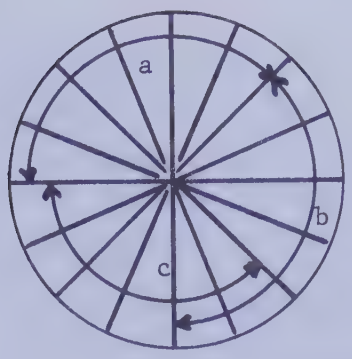
a) en tenant compte de la position instantanée du disque, avant le lancement du S10, on peut déterminer par quelle file doit commencer l'exécution du transfert. On gagne ainsi en moyenne $\frac{1}{2}$ tour de disque (20ms) par transfert.

b) on peut gagner une fraction de tour de disque ($\frac{1}{8}$ ou $\frac{1}{4}$) en changeant l'ordre des deux dernières pages à transférer lorsque les deux conditions suivantes sont remplies simultanément (Voir figure N) :

- il y a recourvrement partiel entre la dernière et l'avant dernière page
- il n'y a pas recouvrement entre l'avant dernière et l'antépénultième

On dit qu'il y a recouvrement entre deux pages quand elles contiennent un ou plusieurs secteurs de même numéro.

gain de $\frac{1}{8}$ de tour



gain de $\frac{1}{4}$ de tour

a : antépénultième
b : avant dernière
c : dernière

On permute b et c

1.4.6.4 Utilisation des 3 disques

A chacun des 3 disques, on associe deux files d'attente de commandes, la file A et la file B. Chacune de ces file contient des sous files F0, F1...F7. Quand une requète arrive, elle est analysée et chaque commande de transfert est dirigée vers le disque concerné où l'une des files reçoit les demandes de vidage et les chargements inconditionnels tandis que l'autre reçoit les chargements conditionnels.

Comme il n'y a qu'un controleur, on ne peut faire de transfert qu'avec un seul disque à la fois. On sert les 3 disques successivement. Quand un disque a été choisi, on prend les commandes dans la file des vidages et des chargements inconditionnels et on prépare avec elles un transfert selon l'algorithme d'optimisation décrit plus haut. On laisse pour une exécution suivante, l'autre file des chargements conditionnels ; on est ainsi certain d'exécuter toujours les vidages avant les chargements conditionnels. On permute ensuite le role des files A et B de façon à lancer, la prochaine fois

les vidages et chargements inconditionnels des nouvelles requêtes. Une file élue pour la $(i+1)^{è}$ exécution sur disque contient :

- des demandes inconditionnelles posées pendant la $i^{è}$ exécution
- des demandes conditionnelles posées pendant la $(i-1)^{è}$ exécution.

1.4.7. Acquittement de la requête

La requête d'un usager est en général acquittée en deux temps (si la requête contient des transferts conditionnels) :

a) à la fin du vidage des pages qui vont être écrasées. On inscrit la situation exacte dans la TPP. Les champs FTPP et ATPP sont mis à jour et associent les pages physiques aux articles qu'elles vont contenir. Le bit de transit reste à 1, indiquant que les nouveaux articles sont en cours de chargement.

b) à la fin de l'exécution de la requête. Les bits T sont mis à 0 dans la TPP. Dans la TPV, la protection effective PETPV des articles chargés est mise à jour, en fonction de l'accès requis et de la protection de fichier PFTPV décidée par le programmeur.

La requête de l'allocateur pour vidage préventif est acquittée en une seule fois. Les bits T sont remis à zéro. Comme des pages ont pu être récupérées durant le vidage, on balaie la file QPV et non la table des requêtes. Les champs FTPP et ATPP ne sont pas modifiés car on doit pouvoir récupérer les pages maintenant libres.

Procédure d'acquittement de la requête d'un usager XUS

```
begin
procédure acquittement_intermédiaire ; comment : concerne les pages
                                                 en vidage et les
                                                 chargements incon-
                                                 ditionnels ;
    for  X := 0  step 1  until dernière entrée de la TPD  do
        begin
        P := PTPD(X) ;
        if P ≠  0 then comment : on ne fait rien si la page a été
                                 demandée par un autre usager
            if VP(X) = 1  then  comment : cas de vidage conditionnel
```

```
                    begin
                    if NA(X) = 1  then
                         begin
                         T(P) := 0 ; comment : amorçage d'article ;
                         remise à zéro de la page P
                         end ;
                    FTPP(P) := FTPD(X) ;
                    ATPP(P) := ATPD(X)
                    end
               else T(P) := 0 ; comment : chargement inconditionnel ;
          end ;

procédure acquittement_terminal ; comment  concerne uniquement les pages
                                           en chargement conditionnel ;
     for X := 0  step 1 until  dernière entrée de la TPD do
          begin
          P := PTPD(X) ;
          if (VP(X) = 1)&(NA(X) = 0)  then T(P) := 0 ;
          comment : on ne teste pas si P = 0  car on a alors VP(X) = 0 ;

          end ;

procédure mise-à-jour_TPV ; comment : mise à jour de la TPV de l'usa-
                                      ger XUS ;
     for X:=0  step 1  until dernière entrée de la TPD  do
          begin
          PTPV(VTPD(X)) := PTPD(X) ;
          PE(VTPD(X)) := PETPD(X)
          end ;
comment :  NCC et OUT sont des compteurs de chargement conditionnels
           et de vidages ;
if NCC = 0  then  comment : pas de chargement conditionnel, mais peut
                            être des vidages s'il y a des amorçages
```

```
     begin
     acquittement-intermédiaire ;
     comment : il n'y a pas d'acquittement terminal dans ce cas ;
     mise-à-jour-TPV ;
     D(S:REQ(XUS)) ;
     comment : activation des processus bloqués pour défaut de page
     end
else
     if OUT ≠ 0 then  comment : chargement conditionnel et vidages,
                                premier temps de l'acquittement
          begin
          acquittement-intermédiaire ;
          OUT := 0
          end
     else
          begin
          acquittement-terminal ;
          comment : deuxième temps de l'acquittement ;
          mise-à-jour-TPV ;
          D(S:REQ(XUS)) ;
          comment : activation des processus bloqués pour défaut de page
          end ;
end
```

Procédure d'acquittement d'une requête de l'allocateur pour vidage préventif

```
begin
P := premier de la file QPV ;
repeat
     begin
     T(P) := 0 ;
     E(P) := 0 ;
     P := SUC(P)
     end
until  P = tête de la file QPV ;
transférer la file QPV dans la file QPL ;
NPL := NPL + NPV ;
NPV := 0
end
```

1.4.8. Libération des pages

Cette opération a lieu au départ d'un usager de la file C, à la fermeture ou à la destruction d'un fichier

a) Au départ d'un usager de la file C, on doit considérer les fichiers privés et les fichiers partagés.

Pour chaque fichier privé F, les pages de QPI(F) rejoignent la queue de QPL et les pages de QPE(F) vont en queue de QPQL. Dans les deux cas on fait A := 0 pour chaque page.

Pour chaque fichier partagé F, on décrémente NUSC, le compteur des usagers courants qui l'utilisent. Si NUSC = 0, on procède comme pour le fichier privé. Sinon, on ne fait rien.

b) A la fermeture d'un fichier, on décrémente NUSC et on procède comme en a).

c) A la destruction d'un fichier, on remet à zéro le champ FTPP et toutes les pages du fichier (écrites et intactes) rejoignent la file QPL. Pour chaque page A := E:= 0.

CHAPITRE 2

COMMUNICATION ENTRE L'UTILISATEUR CONVERSATIONNEL ET LE SYSTEME

2.1 Généralités

La communication entre l'utilisateur conversationnel et le système se déroule par l'intermédiaire du téléimprimeur associé à chaque usager.

Quand un nouvel utilisateur entre dans le système, un certain nombre d'opérations sont à réaliser avant qu'il puisse travailler

- identifier le client et vérifier qu'il a le droit d'utiliser le système
- associer un usager à ce client, pour qu'il puisse utiliser le système et participer au partage des ressources
- lire et interpréter une ou plusieurs phrases du langage de commande permettant de lancer l'exécution d'un programme

En cours de session, l'usager fait appel au moniteur pour exécuter un certain nombre de fonctions. Citons en particulier la gestion de fichiers et l'aide à la mise au point.

Quand un client quitte le système, il convient :

- de fermer tous les fichiers qu'il veut garder et de détruire tous les autres
- de supprimer l'usager qui lui était associé.

Ces diverses opérations pourraient être exécutées par des processus spécialisés du moniteur travaillant en adresses physiques. Il nous est apparu plus souple de les inclure dans un processus particulier de l'usager, le processus PREMIER. En effet :

- la procédure associée au processus PREMIER peut être adaptée au type de chaque usager. Ainsi, les processus PREMIERS du travail en temps réel, du travail de fond ou des usagers conversationnels sont tous différents.

- les processus PREMIERS des usagers conversationnels peuvent être réentrants

- toutes les fonctions assurées par un processus PREMIER sont interruptibles et soumises aux algorithmes généraux de l'allocation des ressources.

Toutefois le processus PREMIER jouit de possibilités d'exécution supérieures à celles des autres processus de l'usager, car il doit accéder aux tables du moniteur.

Enfin, le processus PREMIER est créé et détruit en même temps que l'usager auquel il appartient. On peut donc considérer qu'il y a équivalence entre les notions d'usager et de processus PREMIER.

2.2 Admission et départ d'un utilisateur conversationnel

2.2.1 Admission

L'admission d'un utilisateur est déclenchée par l'enfoncement de la touche appel système (AS) sur un téléimprimeur non encore en session.

On dit qu'un client (et un usager) est en session entre le moment où il a fait l'appel système d'admission et son départ du système.

L'admission d'un utilisateur nécessite des actions de deux types :

- une conversation avec l'utilisateur pour l'identifier et vérifier qu'il a bien le droit d'utiliser le système

- des modifications des tables du système pour y introduire le nouvel usager.

La première action doit se faire au rythme de l'usager et nécessite que le code qui l'exécute soit réentrant. En effet, un usager qui essaie d'entrer dans le système et qui met un certain temps pour cela, ne doit pas empêcher l'admission d'autres usagers. Cette action est donc faite par le processus premier.

La seconde action doit être faite par un module unique, car c'est une section critique du système. Elle concerne des tables du moniteur accessibles par les adresses physiques. Elle est réalisée par un processus du moniteur appelé initiateur.

Pour réaliser l'admission, on procède alors en créant un usager que l'on associe au téléimprimeur, origine de l'appel système. On crée par là même un processus PREMIER qui est chargé de vérifier que le client peut entrer en session. Si ce n'est pas le cas, l'usager est détruit.

Nous pouvons maintenant analyser les actions faites pour l'admission.

Le processus INITIATEUR est activé par le module de gestion des téléimprimeurs qui détecte un AS sur un téléimprimeur hors session :

- il crée un nouvel usager, X, qu'il associe au téléimprimeur origine de l'AS.
- il lui associe une catégorie initiale pour l'allocation des ressources.
- il lui alloue une page de garde, page semi-résidente, qui a pour rôle de servir d'intermédiaire entre le moniteur et les processus de l'usager (en particulier le processus PREMIER). A ce titre cette page de garde (PDG) contient :
  - un tampon d'entrée-sortie utilisé par le téléimprimeur
  - la TPV dans une première version d'ESOPE (nota : l'allocateur est codé de façon à pouvoir vider sur disque la TPV dans une version ultérieure), la TPD et la TFU de l'usager.
  - diverses variables accessibles en adresses physiques par le moniteur et en adresses virtuelles par le processus PREMIER.
- Il met le nouvel usager dans l'état oisif
- Il initialise la TPV et la TFU de l'usager de telle façon que le code du processus PREMIER lui soit accessible. On réalise donc l'équivalent de l'ouverture et du couplage du fichier code et du fichier de travail du processus PREMIER. La page de garde est associée à l'entrée 255 de la TPV.
- Il active le processus PREMIER de l'usager crée, par V(S:PREM(X))

Nota

1. L'allocation d'une PDG se fait en dehors du cycle normal de l'allocation. Pour éviter le blocage, il faut que NDISP soit positif. Par ailleurs, la page allouée pour la PDG doit être prise dans la file des pages libres et non dans la file des pages quasi-libres car on ne lance pas de requête de page pour le moniteur. Si l'une de ces conditions (NDISP > 0, NPL > 0) n'est pas satisfaite, l'INITIATEUR est bloqué derrière un sémaphore particulier S:PL. Lorsque, à la libération de page, NPL devient positif, la primitive D(S:PL) est actionnée et libère l'INITIATEUR s'il est bloqué.

2. On constate que le processus PREMIER n'est pas crée par l'INITITATEUR. En effet, la primitive de création de processus est telle qu'elle ne permet de création qu'à l'intérieur d'un usager. Le processus INITIATEUR, du moniteur, ne peut créer de processus chez un usager. Les processus PREMIERS sont crées actifs à la génération du système. L'usager qui est créé est dans l'état oisif avec un processus actif. Le premier passage dans le processus CHANGEMENT détecte ce processus actif et fait entrer l'usager dans la file prêt des usagers.

Le processus PREMIER

Les conditions d'admission d'un client sont les suivantes :

- son nom figure dans la table des clients
- le mot de passe fourni est celui correspondant à ce nom
- il n'y a pas déjà en session un client portant ce nom

Un client a droit à deux tentatives supplémentaires en cas de refus. Une fois l'usager admis :

- certaines tables du système sont mises à jour, en particulier la correspondance numéro d'usager, numéro de client.
- le catalogue des fichiers du client est transféré de DIMAS sur RAD
- un message d'admission est émis sur le téléimprimeur de l'usager ('USAGER UNTEL ADMIS')
- le processus premier passe à son état de référence (avec émission du message 'ESOPE'), en attente d'une commande.

2.2.2 Départ

Le départ d'un usager est la conséquence d'une commande particulière interprétée par le processus PREMIER. Les opérations associées au départ sont réalisées par le processus PREMIER, puis par un processus du moniteur, le BALAYEUR.

Le processus PREMIER :

- détruit tous les autres processus de l'usager et efface tous les sémaphores créés par lui.
- ferme tous les fichiers de l'usager, sauf celui du processus PREMIER. Si l'usager a des fichiers de travail, ils sont détruits.
- demande au client quels sont les fichiers qu'il faut conserver (et sauver sur DIMAS). Détruit tous les autres.
- conserve le catalogue des fichiers du client en le transférant sur DIMAS.
- sort diverses statistiques et éléments comptables
- coupe l'alimentation électrique du téléimprimeur.
- active le processus BALAYEUR après lui avoir passé un message indiquant le numéro de terminal.

Le processus BALAYEUR :

- ote l'usager de la file des courants (c'était l'usager élu) et libère toutes ses pages.
- rend la page de garde à la file des pages libres (NDISP := NDISP + 1 ; NPL := NPL + 1 ;)
- modifie les tables du système pour indiquer que l'usager n'existe plus. En particulier le téléimprimeur n'est plus associé à aucun usager.
- active le processus CHANGEMENT pour élire un autre usager.

2.3. Le langage de commande

L'usager communique avec le système et lui demande des services par un langage de commande.

Le langage de commande doit permettre :

- l'admission et le départ d'un usager

- l'utilisation des programmes du système (appel de traducteurs partageables, exécution de programmes privés).

- la gestion des données (création, destruction, ouverture, fermeture de fichiers, édition, recopie).

- la fourniture de services divers (administration du système, mesures, statistiques).

Les commandes ne sont acceptées que lorsque le système est dans un état de référence particulier. Ceci est indiqué à l'utilisateur par la sortie du message 'ESOPE'.

2.3.1 Syntaxe du langage

Nous avons adopté une syntaxe à base de mots clés permettant de mettre les paramètres d'une commande dans un ordre quelconque, et éventuellement d'en omettre

< commande > ::= < nom de commande > | < nom de commande > < blanc > < liste d'argument >

< liste d'arguments > ::= < argument > | < liste d'arguments >,<argument>

< argument > ::= < argument simple > | < argument à clé >

< argument simple > ::= < identifieur > | < nombre >

< argument à clé > ::= (< clé >,< liste d'arguments simples >) |(< clé >)

< liste d'arguments simples > ::= < argument simple > | < liste d'arguments simples >,< argument simple >

L'ordre est significatif pour les arguments simples ; il ne l'est pas pour les arguments à clé.

Exemple : *ESPION
*CPL1 TOTO
*COPIER TOTO, LULU
*MODIFIER TOTO,(NOM,LULU), (AUT,DUPOND,O,DURAND,1)

2.3.2 Admission. Départ d'un usager

Nous avons vu l'effet de ces commandes au chapitre précédent :

- ADMISSION : obtenue par l'enfoncement de touche AS d'un télétype hors session.

-*DEPART : met en route la séquence des actions qui otent l'usager du système.

2.3.3 Gestion des fichiers du système

Il n'y a pas de commande explicite de création de fichier. La création de fichier est faite automatiquement lorsque le nom apparait pour le première fois comme argument d'une commande

*DETRUIRE : ferme et détruit le fichier indiqué. S'il est ouvert ou associé par un autre usager, la destruction est différée jusqu'au départ de cet usager.

*VIDER : place le repère de caractère courant en tête de fichier.

*RECOPIER : recopie un fichier à la suite d'un autre.

*CATALOGUER : imprime selon les options, le catalogue du système ou le catalogue de l'usager.

*MODIFIER : change les caractéristique d'un fichier. (type, accès permis, liste des utilisateurs autorisés...).

*IMPRIMER : sort , en différé, sur l'imprimante le contenu du fichier indiqué.

*SAUVER : commande le transfert d'un fichier interne vers la DIMAS.

*RAPPELER : commande le transfert d'un fichier externe vers le RAD.

Toute ces commandes sont directement interprétées par le processus PREMIER.

2.3.4 Gestion des programmes

Les commandes de ce type demandent l'exécution d'un fichier du système (commande précédée d'un astérisque) ou d'un fichier de l'usager. Dans ce dernier cas, le nom de commande doit être le nom d'un fichier de l'usager.

Une telle commande se déroule en plusieurs phases :

- analyse de la commande
- chargement du fichier, qui doit être un module de chargement
- création d'un processus associé à ce fichier. On l'appelle ici TRADUCTEUR
- exécution de ce processus TRADUCTEUR. Au cours de cette exécution, des échanges conversationnels sont possibles. Ils peuvent être administrés à l'aide d'un langage propre à ce traducteur.
- retour au processus premier en fin de commande.

L'analyse de la commande se fait en deux étapes :

- l'analyse du nom de commande est effectuée par le processus PREMIER. Si le nom commence par le caractère * , on le recherche dans la table des commandes. Sinon on regarde s'il figure dans le catalogue de l'usager. En cas d'insuccès de la recherche dans l'un ou l'autre cas, la commande est illégale.

- l'analyse des paramètres de la commande est à la charge du processus TRADUCTEUR destinataire. Elle sera donc faite seulement après la création de ce dernier. Toutefois le processus PREMIER peut faire une première analyse destinée à détecter les erreurs de syntaxe et à éviter d'inutiles mouvements de fichiers.

Le chargement du fichier fait appel au chargeur, procédure fonctionnant dans l'espace virtuel et implantée à une adresse virtuelle fixe et connue. Les paramètres à passer au chargeur sont

- le FV (entrée de la TFU) du module de chargement sur lequel il doit opérer
- une option concernant le traitement des données (protection 0)
- un niveau de sévérité acceptable
- une adresse de départ.

Le chargeur vérifie d'abord que le fichier indiqué est bien un module de chargement. Ensuite, il vérifie que le niveau de sévérité figurant dans le module est compatible avec le niveau acceptable. Enfin, le couplage proprement dit est exécuté :

- les zones de protection 1 et 2 (code, littéraux) sont couplées aux pages virtuelles indiquées, avec les protections 1 et 2
- pour les zones de protection 3 (zones d'"overlay" où le traducteur lui même couplera des pages) on ne se contente de placer 3 dans les zones PE et PF de la TPV et dans les bits de protection de la mémoire topographique.
- en ce qui concerne les zones de protection 0 (zones de données), il y a deux possibilités
  - ce sont des données d'un fichier privé : les articles correspondants sont couplés aux pages virtuelles indiquées avec la protection 0
  - ce sont des données d'un fichier dont le code est partagé (traducteur du système) : on crée un fichier de travail qui est initialisé avec les valeurs trouvées dans les articles de protection 0 du module de chargement.

On désigne au chargeur l'une de ces deux options. La seconde est obligatoire pour les traducteurs partagés du système.

Après le chargement, le processus premier crée un processus TRADUCTEUR en indiquant comme adresse de départ celle qui est indiquée dans le module de chargement. Le traducteur est crée actif ; il devient élu lorsque le processus PREMIER passe dans sa position d'attente en affectuant P(S:PREM)

Le retour du traducteur au processus PREMIER se fait par une primitive spéciale E:EXIT qui place un message pour le processus PREMIER et l'appelle par V(S:PREM). Celui ci revient à son état de référence (envoi du message 'ESOPE' et attente d'une commande.)

Remarques

1. On peut toujours revenir au processus PREMIER en cours d'exécution du TRADUCTEUR grâce à la touche appel système. Le mécanisme suivant a été codé :

- en appuyant une fois sur la touche AS, on obtient la mise à 1 d'un indicateur BLNAS associé au terminal. Cet indicateur peut être utilisé par un traducteur pour répondre à l'AS. Une telle utilisation remet BLNAS à zéro.

- si on appuie sur la touche AS alors que BLNAS est encore à 1, un message est envoyé au processus PREMIER et ce dernier est activé par une primitive D(S:PREM). BLNAS est remis à zéro.
Si le processu PREMIER est en position d'attente derrière P(S:PREM), il interrompt tous les processus de l'usager, envoie un message 'ARRET POUR APPEL SYSTEME' et passe dans l'état de référence (envoi du message 'ESOPE' et attente de commande).

2. On ne détruit pas le TRADUCTEUR lorsqu'on revient au processus PREMIER par E:EXIT ou par un double appel système. On ne le fait que lorsqu'une nouvelle commande le nécessitera pour créer un nouveau TRADUCTEUR.

Les commandes qui demandent l'exécution d'un fichier du système sont les suivantes :

*RELIER : les fichiers indiqués comme contenant des modules objet sont reliés en un seul module de chargement qui contient des pages de code binaire absolu exécutable, des pages de données et des pages de manoeuvre pour l'"overlay". Le programme appelé par cette commande est nommée relieur.

*EDITER : éditeur de texte conversationnel

*CPL1 : compilateur et interpréteur conversationnel du langage PL1

*LP 10070 : compilateur non conversationnel du langage LP 10070. Ce compilateur fournit un module objet dans un fichier indiqué

*CLP : compilateur conversationnel du langage LP 10070

2.3.5 Services divers

Un certain nombre de services peuvent être demandés au processus PREMIER :

*MARCHE : s'utilise après le double AS et indique qu'il faut continuer la commande interrompue

* * * : la suite de cette ligne n'est qu'un commentaire

* POINT : permet d'accéder aux processus, aux sémaphores et à l'espace virtuel de l'usager. Grâce à elle on peut :

- connaitre la file de sémaphore où se trouve un processus
- connaitre la valeur d'un sémaphore de l'usager
- lire et modifier le contenu des registres d'un processus de l'usager
- lire et modifier le contenu d'une cellule de la mémoire virtuelle

*ESPION : donne la liste des clients en session et indique sur quels téléimprimeurs ils sont

*CODE : change le code de l'usager.

*HEURE : fournit l'heure

*STATISTIQUES : fournit des statistiques sur le comportement du système.

2.4 Signalement des erreurs

Nous avons prévu deux types de réactions à l'apparition d'un déroutement ne concernant pas le système, mais l'usager élu. Toutes deux renvoient à l'usager. Dans le permier cas, ou déroutement standart, l'utilisateur est averti du déroutement à son téléimprimeur. Dans le second cas, ou déroutement non standart, on lance l'exécution d'une procédure écrite par l'usager et donnant une action spécifique.

2.4.1 Déroutement standart

Ce cas standart fait appel au processus PREMIER. Le sous programme de déroutement place un message dans la page de garde de l'usager pour indiquer l'adresse du déroutement, le nom du processus dérouté, la cause et le numéro du déroutement, puis il active le processus PREMIER.

Le processus PREMIER interrompt alors tous les processus de l'usager en les plaçant derrière le sémaphore S:ERREUR(XUS), sort un message sur le téléimprimeur, indiquant les résultats transmis et passe dans l'état de référence (message 'ESOPE' et attente d'une commande).

2.4.2 Déroutement non standart

Dans ce cas l'adresse de la procédure à exécuter a été placée au préalable dans la page de garde où elle remplace l'adresse du sous programme de déroutement standart. Ainsi en cas de déroutement, le branchement vers la procédure non standart se fait automatiquement.

Cette méthode est particulièrement adaptée aux traducteurs qui traitent eux mêmes les déroutements et envoient leurs propres messages d'erreur à l'utilisateur sans pour autant revenir au processus PREMIER.

2.5 Le client ESOPE

Un certain nombre d'opérations touchant à l'administration du système sont réservées à un client privilégié nommé ESOPE. Ces opérations sont les suivantes :

- gestion de la table des clients
- exclusion d'un usager, neutralisation d'un téléimprimeur
- adjonction et suppression de traducteurs

2.5.1 Gestion de la table des clients

Les opérations possibles sont :
- ajouter un nouveau client
- supprimer un client
- lire le mot de passe d'un client
- lire et modifier la clé d'autorisation d'un client

A chaque client est associé une clé d'autorisation lui donnant certains droits. La signification de la clé est la suivante.

Bit n°	Action (1 = autorisée ; 0 = interdite)
0	Ajouter/supprimer un client
1	Exclure un usager
2	Lire le mot de passe d'un client
3	Introduire un nouveau traducteur
4	Modifier la clé d'autorisation d'un client
5	Devenir meneur du Temps Réel

Nota : tout client (et lui seul) peut modifier son mot de passe.

2.5.2 Exclusion d'un usager

L'intérêt de cette opération est de sortir du système un usager qui pour une raison quelconque ne peut sortir lui-même (panne de téléimprimeur, erreur dans le processus PREMIER pendant sa phase de mise au point, mauvais usage du test sur le BLNAS d'appel système...). On peut ainsi libérer les pages occupées par l'usager en mémoire centrale et sur RAD.

Le principe de l'exclusion consiste à simuler une commande * DEPART exécutée par le processus PREMIER de l'usager à exclure, ou ce qui revient au même, à activer ce processus à une adresse déterminée. Or, à l'instant de l'exclusion ce processus peut se trouver dans un état quelconque, et, en particulier, bloqué derrière un sémaphore pour une durée infinie (cas de blocage derrière un sémaphore associé à un téléimprimeur en panne). On devra donc

- arrêter l'entrée-sortie éventuelle en cours sur le téléimprimeur de l'usager exclus
- déterminer le sémaphore derrière lequel se trouve bloqué le processu PREMIER :
  - si c'est S:REQ, ne rien faire, car il sera débloqué à l'acquittement de la requête
  - sinon faire un D sur ce sémaphore. Cette action libèrera tous les processus qu'il bloquait et processus PREMIER, le plus prioritaire sera élu.
- fixer le PSD du processus premier à l'adresse voulue.

2.5.3 Adjonction et suppression de traducteurs

Si l'on désire introduire un nouveau traducteur on doit :

- constituer, grâce au relieur, le module de chargement composant le traducteur
- introduire ce module dans le catalogue des fichiers du client ESOPE
- introduire dans la table des commandes le nom de la commande correspondante ainsi qu'un pointeur vers le fichier associé

Cette opération se fait par une commande

*DEFINIR TOTO,(NOM,TRUC)

où TOTO doit être le nom d'un fichier figurant dans le catalogue du client ESOPE. Cette commande introduit une nouvelle commande de nom *TRUC, qui déclenche l'exécution du fichier TOTO.

L'opération inverse (suppression de la commande) retire le nom de la table des commandes. Elle n'a aucune action sur le fichier.

2.6 Processus PREMIER de l'usager "temps réel"

2.6.1 Admission

Un client desirant lancer un travail en temps réel est admis dans le système comme un usager conversationnel ordinaire. Une fois admis (message 'ESOPE'), il peut demander à devenir meneur de l'usager "temps réel". C'est possible si :

- il a l'autorisation (dans sa clé) de la faire
- il n'y a pas déjà un usager temps réel dans le système.

L'effet de cette commande est de charger dans la mémoire virtuelle de l'usager (par couplage) les parties spécifiques du processus PREMIER du "temps réel".

2.6.2 Allocation et libération de téléimprimeurs

Après la commande de passage en temps réel, le processus PREMIER de l'usager temps réel se place dans son état de référence, où il sort un message sur le terminal du meneur ('ESOPE TEMPS REEL') et attend une commande.

Le meneur peut alors demander l'allocation de téléimprimeurs, un par un, en spécifiant le numéro de téléimprimeur physique désiré. Les numéros internes sont obligatoirement 1, 2, 3, ... (<32). Si un téléimprimeur demandé est occupé, un message est envoyé au meneur et l'allocation n'est pas faite. A chaque télétype, on associe une page de garde. La page de garde associée au téléimprimeur de numéro interne i est la page virtuelle numéro 255-i. Par convention, le numéro interne 0 correspond au terminal du meneur.

L'appel système, tel qu'il a été introduit plus haut, concerne uniquement le terminal du meneur.

La libération des terminaux est faite depuis le téléimprimeur du meneur. La commande * DEPART n'est également valable que depuis le terminal du meneur. Elle comporte la libération automatique des terminaux esclaves non encore libérés. Les autres opérations sont les mêmes que pour un usager conversationnel ordinaire.

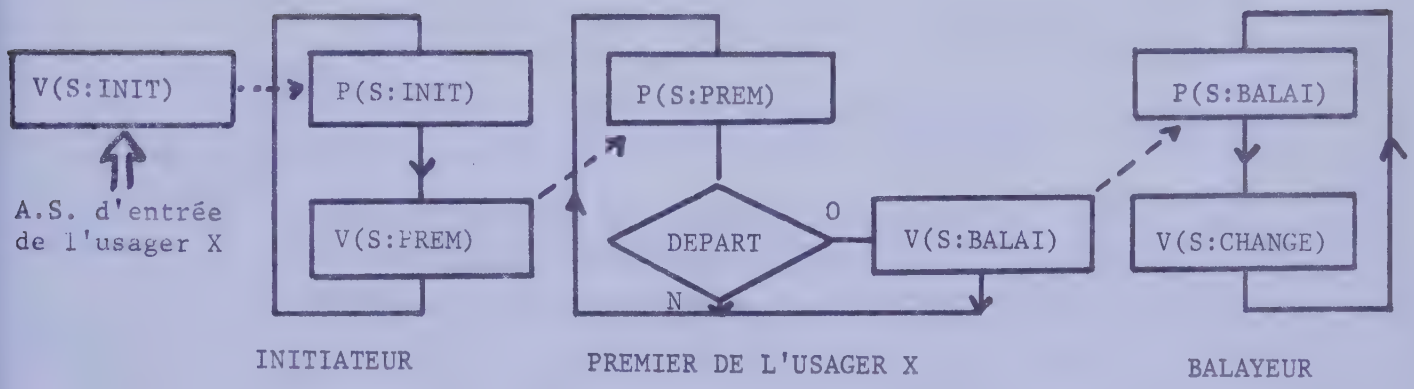
2.7. Les processus de la communication usager-système

2.7.1 Coopération à l'admission et au départ

```
INITIATEUR :
          begin
    A :   P(S:INIT) ;
          Allouer un numéro d'usager, soit X ;
          Associer X au téléimprimeur  origine de l'appel système ;
          Donner à X une catégorie initiale ;
          while (NPDISP ⩽ 0) | (NPL ⩽ 0) do P(S:PL) ;
          allouer à X une page de garde ;
          initialiser la TPV et la TFU de l'usager X ;
          état (X) := oisif ;
          mettre X dans la queue des usagers oisifs ; comment comme son
          processus PREMIER est actif, l'usager X passera prêt à la
          prochaine activation de CHANGEMENT ;
          V(S:PREM) ;
          goto A
          end ;

PREMIER : begin
     A :  P(S:PREM) ;
          X:=usager élu ;
          for i := 1  step 1  until 3  do
               begin
               imprimer (VOTRE NOM) ;
               if le nom figure dans la table des clients then
                    if le nom n'est pas celui d'un client en session then
                         begin
                         imprimer (VOTRE CODE) ;
                         if le mot de passe est correct then
                              begin
                              imprimer (USAGER ADMIS) ;
                              goto  ETAT DE REFERENCE
                              end
                         end
               end ;
```

```
DEPART :  détruire tous les processus et effacer tous les sémaphores
          de l'usager ;
          fermer tous les fichiers de l'usager sauf celui du processus
          PREMIER ;
          détruire les fichiers de travail de l'usager ;
          ranger les fichiers à conserver, détruire les autres ;
          imprimer (USAGER  SORTI) ;
          V(S:BALAI) ;
          goto A ;

ETAT DE REFERENCE : imprimer (ESOPE) ;
          attendre les commandes ; comment : reste du code du processus
                                             PREMIER ;
          ...
          end ;

BALAYEUR : begin
     A :  P(S:BALAI) ;
          X := usager élu ;
          ôter l'usager X de la file courante ;
          rendre la page de garde de X à la liste des pages libres ;
          dissocier X et le téléimprimeur ;
          V(S:CHANGE) ;
          goto A
          end ;
```

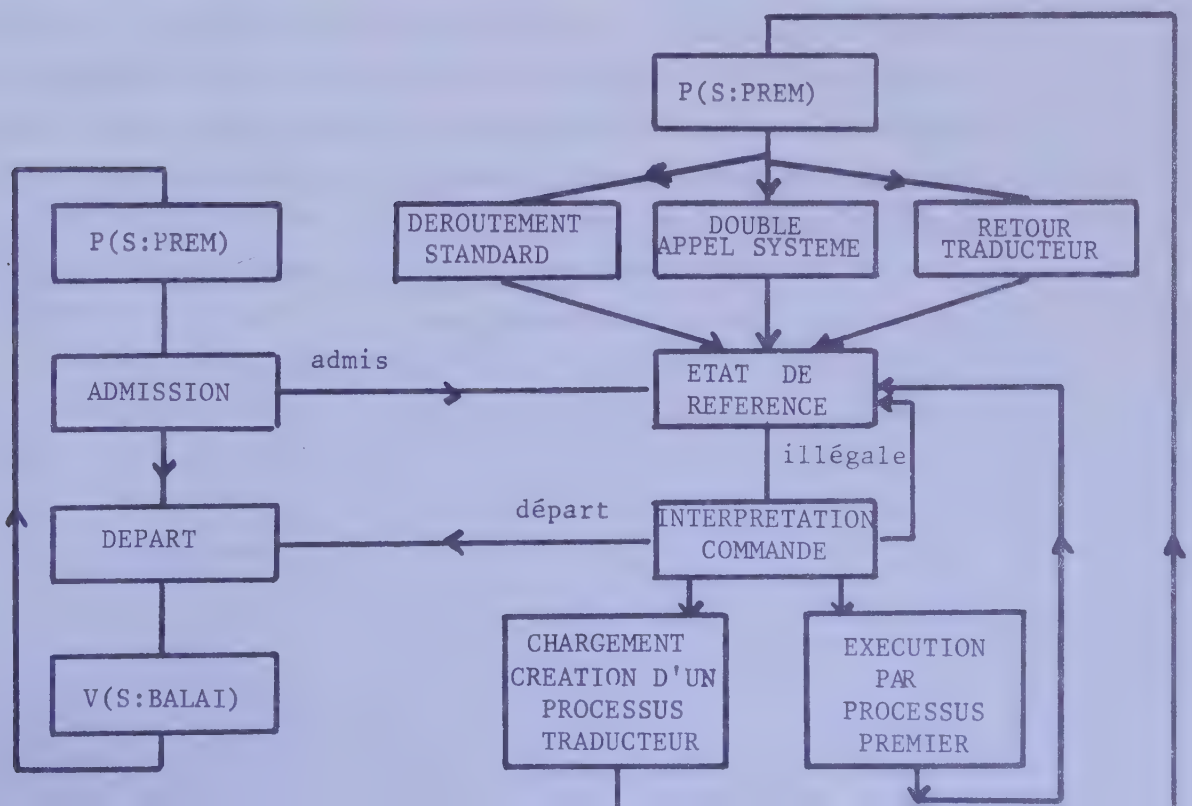
La figure ci-dessous résume cette coopération. Le sémaphore S:PL n'y figure pas pour simplifier la figure.



2.7.2 Le processus PREMIER

2.7.2.1 Structure

La structure d'ensemble du processus PREMIER des usagers conversationnels est la suivante :



Rappelons que l'état de référence est celui où le processus PREMIER imprime ESOPE et attend une commande de l'utilisateur.

Remarque : La boucle P(S:PREM) ; admission ; départ ; V(S:BALAI) n'est parcourue qu'une seule fois pendant l'existence de l'usager. La primitive P(S:PREM) est donc inutile. De même le V(S:PREM) qui figure dans l'initiateur. Toutefois comme le processus PREMIER n'est ni créé ni détruit, la boucle doit subsister pour permettre de multiples utilisations du même numéro d'usager, donc du même processus PREMIER.

2.7.2.2 Pages privilégiées

On a vu que le processus PREMIER avait à communiquer avec les processus du moniteur et avec le module de gestion des téléimprimeurs. Ces divers éléments travaillent en adresses physiques, tandis que le processus PREMIER utilise l'adressage virtuel.

Cette communication est règlée de la façon suivante :

- la page 0 physique et toutes les pages 0 virtuelles sont "collées", c'est à dire que toutes les pages 0 des mémoires virtuelles ont comme page physique associée la page 0 physique. Cette page 0 permet alors à tous les usagers de communiquer entre eux ou avec le moniteur.
- la page 1 physique est collée à toutes les pages 1 virtuelles
- la page de garde PDG est la page virtuelle 255 pour chaque usager. Un pointeur, rangé dans la page 0, contient le numéro de page physique associée à la PDG qui, rappelons le, est semi résidente. Cette PDG peut servir de communication entre le moniteur (ou le module de gestion des téléimprimeurs) et l'usager.

Pour éviter qu'un processus quelconque d'un usager n'aille écrire dans les pages 0, 1 ou la PDG, en y accédant en virtuel, on les protège à l'aide d'un verrou contre l'écriture physique. Les pages 0 et 1 et la PDG ont le verrou 2. Seul le processus PREMIER a la clé 2 tandis que les autres processus de l'usager ont la clé 1. Par ailleurs les processus du moniteur ont la clé 0 (clé passe partout) qui leur permet d'accéder à toutes les pages.

La page 0 contient en dehors des emplacements réservés pour les interruptions et les déroutements :

- une zone de communication entre INITIATEUR, BALAYEUR et processus PREMIER

- des variables d'intérêt général (pointeur vers l'usager élu...)

- une table des usagers téléimprimeurs

- une partie de la table des usagers (en particulier l'état d'un usager)

- des compteurs de mesure

La page 1 contient :

- une partie de la TFO

- la table des catalogues sur RAD (utilisée par la gestion des fichiers)

- une zone de communication entre l'usager PTT et les autres usagers (boite aux lettres)

La page de garde de chaque usager comprend :

- des tampons pour le téléimprimeur associé à cet usager

- la table des pages virtuelles de l'usager (TPV)

- la table des pages demandées par l'usager (TPR)

- la table des fichiers de l'usager (TFU)

- l'identité du client associé à l'usager (nom, clé)

- des compteurs de mesure.

2.7.2.3 Partie résidente

Le processus PREMIER, étant le plus prioritaire des processus de l'usager est activé le premier dès qu'il est débloqué. Toutefois la primauté sur les autres processus ne lui est réellement assurée que s'il ne fait pas de défaut de page lors de son activation (voir § 5.2.1.1.). Le seul cas où cela peut se produire est la sortie de la position d'attente sur P(S:PREM). On a vu qu'on pouvait avoir un délai d'activation du processus PREMIER égal au temps de résidence ou même infini (cas d'un processus effectuant une boucle dans les registres et ne faisant donc pas de défaut de page).

Pour éviter cette situation, on laisse résidente la partie du code du processus PREMIER correspondant à la sortie de la position d'attente, c'est à dire :

```
P(S:PREM) ;
INTERROMPRE
```

Ainsi, dans tous les cas, le processus PREMIER sera activé dès son déblocage et sa première action sera d'interrompre tous les autres processus de l'usager.

CHAPITRE 3

LA GESTION DES PERIPHERIQUES

3.1 Introduction

Le système d'entrées sorties (SES) d'ESOPE est conçu comme une extension au système conversationnel et non comme un système capable d'accepter de gros travaux de gestion. [1], [2]. Pour la clarté de l'exposé, nous ne présentons que la gestion des imprimantes et du lecteur de carte et que les mouvements d'information entre un de ces périphériques et un fichier interne. Cette gestion fait intervenir la console de l'opérateur.

La conception du système repose sur les trois idées suivantes :

1) la réalisation de la partie conversationnelle d'ESOPE a imposé le développement de mécanismes appropriés (gestion des processus, des fichiers, allocation de ressources). Pour la réalisation du SES, il faut autant que possible tirer parti des mécanismes existants.

2) le SES doit être efficace. Nous entendons par là que la structure globale du SES doit permettre d'exploiter au mieux le matériel (si deux transferts peuvent se dérouler en parallélisme réel, il n'y a pas de raison de s'en priver) ; l'amélioration des performances par optimisation locale du code objet ne doit venir qu'en second lieu.

3) le SES doit être aussi indépendant que possible du reste du système et avoir un interface bien défini. Il ne s'agit pas ici d'un voeu, mais d'une exigence, car la réalisation du système conversationnel a effectivement précédé celle du système d'entrées-sorties.

3.2 Les mécanismes utilisés

3.2.1 Instructions d'entrée-sortie du CII 10070 [3]

Le 10070 dispose de 5 instructions d'entrée-sortie fonctionnant en mode maître :

SIO (P) : lance une opération d'entrée-sortie avec le périphérique P

HIO (P) : arrête toute opération avec le périphérique P. Si celui-ci a émis une interruption, elle est annulée.

TIO (P)
TDV (P) } : fournit l'état du périphérique P, n'a aucune action sur P.

AIO : indique si un signal d'interruption a été émis par un périphérique. Dans ce cas, fournit le nom et l'état du périphérique émetteur le plus prioritaire et enlève le signal que celui-ci a déposé.

Pour chaque instruction, l'unité centrale fournit une réponse (par le code condition et par registres) :

- réalisation ou non de l'opération demandée,
- état du périphérique mis en jeu.

Ces instructions de l'unité centrale sont destinées à une unité d'échange qui se charge de l'opération d'entrée-sortie elle-même. L'unité centrale n'attend pas la fin de celle-ci et poursuit en séquence. L'opération d'entrée-sortie est décrite par un programme-canal formé d'une suite de commandes que doit exécuter l'unité d'échange.

Chaque commande CCW est formée :

- d'un ordre définissant une action pour le périphérique (lire, écrire, rebobiner...) ou une rupture de séquence dans le programme canal,
- et d'indicateurs précisant les actions à exécuter à la fin de l'ordre (continuer en séquence avec la commande suivante, envoyer une interruption de fin de transfert) ou en cas d'erreur (envoyer une interruption ou non).

L'instruction SIO a comme paramètre l'adresse de la première commande à exécuter. Les instructions HIO et TIO fournissent comme complément de réponse, l'adresse de la commande en cours au moment de l'instruction.

Remarques 1. L'état d'un périphérique peut être connue à tout moment par l'unité centrale. Tout changement d'état du périphérique ayant une conséquence sur les transferts d'information peut être signalé par une interruption sauf le passage dans l'état manuel (obtenu en pressant sur le bouton MANUAL). Il faut donc détecter l'état manuel par une interrogation périodique des périphériques

2. Il n'y a pas d'exclusion d'accès au programme canal entre unité centrale et unité d'échange. On ne peut, en toute sécurité, ajouter des commandes à un programme canal en exécution car il faut tenir compte des vitesses relatives d'exécution des processus sur l'unité centrale et sur l'unité d'échange. Or ces vitesses sont aléatoires, à cause des interruptions plus prioritaires (horloges, déroutement, panne secteur.....).

3. Les unités d'échange ont accès à la mémoire centrale physique sans passer par la mémoire topographique. Les commandes et les zones de mémoire qu'elles utilisent doivent donc être résidentes pendant le transfert.

3.2.2 Insuffisance des mécanismes actuels

Dans la version purement conversationnelle d'ESOPE, les pages de mémoire sont soit résidentes, et elles sont alors fixées une fois pour toutes au démarrage du système (et en général non accessibles par l'intermédiaire de la mémoire topographique) soit non-résidentes, c'est-à-dire que leur contenu peut être vidé sur disque pour les besoins de l'allocation de ressources.

Lors de l'implantation du SES,il faut résoudre le problème de l'emplacement des tampons : si on les place dans la zone résidente, comment va-t-on y accéder à partir d'une mémoire virtuelle ? Si on les place en mémoire virtuelle, comment va-t-on les immobiliser en mémoire centrale pendant la durée des transferts ?

Par ailleurs, tous les transferts que nous considérons ici font intervenir à une extrémité des accès à un fichier interne sur RAD. A priori, deux solutions sont envisageables :

1) soit développer un mécanisme d'accès ad hoc, en demandant directement les entrées-sorties au module de gestion des disques,

2) soit utiliser, comme n'importe quel autre usager, le mécanisme du couplage.

La première solution conduit à des modifications du système conversationnel (en plus de la création de nouveaux mécanismes).

Dans la seconde, les sous-programmes chargés de lire les articles de fichier en utilisant le couplage doivent s'exécuter en virtuel et en mode esclave, c'est-à-dire être inclus dans un usager d'LSOPE.

Comme l'autre extrémité du transfert fait intervenir des unités d'échange ne connaissant que des adresses physiques, il faut mettre en place un mécanisme de communication entre la mémoire virtuelle et la mémoire physique.

Nous avons préservé l'emploi du couplage grâce à l'introduction d'un seul mécanisme nouveau, le collage. Il permet, très simplement, d'avoir des tampons résidents et d'y accéder depuis une mémoire virtuelle.

3.2.3 Le collage

C'est dans la TIF qu'apparait le support physique d'un fichier. Ce support est modifié par le système selon les nécessités de l'allocation de mémoire. Si on veut immobiliser une page de fichier en mémoire centrale, c'est dans la TIF qu'il faut le faire apparaître. Cette méthode conserve la généralité du couplage mais présente les inconvénients d'imposer une modification de l'allocateur en introduisant pour les pages non allouées une file de pages immobiles, en plus des files libres et quasi-libres et de conserver le défaut de page, après chaque retour de l'usager à l'état courant, sur des pages que l'on sait être en mémoire centrale.

Pour éviter ces inconvénients, nous avons préféré agir au niveau de la mémoire virtuelle et contrairement à ce qui se passe pour le couplage, associer directement un support à des pages virtuelles privilégiées.

Pour pouvoir retenir des pages virtuelles en mémoire, ou les coller, nous avons partagé la mémoire virtuelle en 3 zones délimitées par deux repères MIN et MAX.

Seules les pages virtuelles v telles que

$$MIN \leq v \leq MAX$$

peuvent être couplées et peuvent demander, en cas de défaut de page, une allocation de page physique. Les autres pages, ou bien ne sont pas accessibles, ou bien ont des pages physiques allouées en permanence. Ainsi pour un usager conversationnel ordinaire, MIN > 1 et MAX < 255 car les pages 0 et 1, collées aux pages physiques 0 et 1, sont partagées entre tous les usagers et le moniteur, et la page 255 (page de garde) est collée à une page physique pendant toute la durée de la session, pour en particulier servir de tampon d'entrée et de sortie des messages avec le téléimprimeur.

Comme conséquence, seules les pages physiques associées aux pages virtuelles entre MIN et MAX sont rendues à l'allocateur (pages libres ou quasi-libres) lorsque l'usager n'est plus courant (voir états des usagers dans [8]). Les pages virtuelles collées ne génèrent jamais de défaut de page.

Deux primitives particulières, allouer-tampon et libérer-tampon, permettent le collage et le décollage des pages virtuelles ; la fixation des bornes MIN et MAX par le processus premier évite un emploi abusif de ces primitives.

Avec ce mécanisme, le passage d'un article de fichier (a,f) de son support à une page physique se programme très simplement en mémoire virtuelle :

soit PV1, PV2 (MIN ≤ PV1 ≤ MAX < PV2) 2 pages virtuelles

coupler (PV1, a, f) ;
allouer-tampon (PV2)
recopier PV1 dans PV2 ;

La primitive allouer-tampon fournit de plus le numéro de la page physique effectivement collée à la page virtuelle ; on peut utiliser ce numéro pour adresser la page à partir d'un processus s'exécutant en adresses physiques.

3.3 Schéma de réalisation du SES

3.3.1 Problèmes de synchronisation

Le schéma connu sous le nom de "producteur-consommateur" permet de programmer efficacement la coopération de deux processus évoluant avec des vitesses quelconques (voir 2e partie § 2.2.6. Voir aussi [4]). Nous l'avons utilisé systématiquement.

On notera :

1. que dans le cas de producteurs et de consommateurs multiples l'accès à la zone de tampons de communication doit être fait dans une section critique.

2. que, dans la pratique, la programmation se complique parce que les deux processus coopèrent pendant un temps fini et qu'il faut programmer l'arrêt, et éventuellement la destruction, des processus producteurs et consommateurs.

3.3.2 Processus mis en jeu

1. Pour le transfert entre un fichier et un périphérique, deux processus interviennent :

- le pilote, qui conduit le périphérique. Il se déroule en mode maître et accède aux adresses physiques,

- le facteur, qui accède au fichier par le mécanisme du couplage. Il se déroule en mode asservi et accède aux adresses virtuelles.

Le mécanisme du collage est utilisé pour définir une zone de tampon d'une page chacun. Chaque tampon est accédé en virtuel par le facteur et en physique par le pilote. Dans le cas d'entrée d'information vers les fichiers, (par exemple, lecteur de cartes), le pilote est producteur et le facteur consommateur. En sortie (par exemple, imprimantes), le facteur produit et le pilote consomme.

2. Deux processus également entrent en jeu pour transmettre les demandes de transfert :

- le facteur qui dans cette relation est toujours consommateur de messages
- l'émetteur qui produit les messages. La nature de ce processus varie selon les périphériques.

Ainsi pour les imprimantes, les émetteurs sont les processus premiers des usagers et les messages sont garés dans un fichier du système - la boite aux lettres. L'émetteur, comme le facteur, est un processus virtuel (utilisant l'adressage virtuel).

Pour le lecteur de carte, l'émetteur est le processus servant associé à la console de l'opérateur. Les seuls messages sont l'ordre de lecture du paquet de cartes et l'ordre d'arrêt de lecture ("avort") qui engendre des opérations V sur des sémaphores. La fin de lecture est normalement lue sur une carte spéciale (!FIN). Le servant est un processus physique (utilisant l'adressage physique).

3. Pour n'avoir qu'une seule mémoire virtuelle à gérer, les facteurs ont été regroupés dans un seul usager, l'usager PTT qui comprend donc les processus virtuels suivants :

- un processus premier
- un facteur de lecture de cartes
- deux facteurs d'impression (un par imprimante).

3.4. Commande et surveillance des périphériques

3.4.1 Fonctions à réaliser

Ce sont les suivantes :

a) lancement du lecteur de carte par l'opérateur,

b) demandes de services à l'opérateur :
corrections d'erreurs et remise en marche du périphérique correspondant, mise en automatique d'un périphérique resté en manuel,

c) détection d'un mauvais fonctionnement des périphériques,

d) interventions sur l'état des périphériques ou le déroulement des transferts : verrouillage d'une imprimante, avorter une impression ou une lecture de cartes.

Ces fonctions sont réalisées par deux processus physiques : servant et chien de garde.

3.4.2 Etats des périphériques

Nous avons introduit diverses variables pour traduire la situation du périphérique dans le SES. Ces variables tiennent compte des interventions de l'opérateur et des modes câblés apparaissant dans les coupleurs des périphériques. Nous distinguons entre transfert d'information, opération élémentaire commandée par un SIO et mouvement d'information, opération globale sur un fichier. Les états considérés sont :

ACTIF : le facteur et le pilote du périphérique sont engagés dans un mouvement d'information. Pour la surveillance du pilote seul, nous utilisons l'état PILACT pour indiquer que le pilote génère des opérations SIO.

AVORT : le facteur et le pilote du périphérique terminent prématurément le mouvement d'information en cours et se placent dans un statut leur permettant d'effectuer le mouvement suivant. Cet arrêt prématuré se fait à la demande de l'opérateur. Lorsque l'arrêt est terminé, le périphérique quitte l'état AVORT.

VERROU : le facteur et le pilote du périphérique sont bloqués ou vont l'être dès la fin du mouvement en cours. Ce blocage, demandé par l'opérateur subsistera jusqu'à une demande de déblocage faite par l'opérateur.

ATTEND : le pilote a détecté une anomalie et l'a signalée à l'opérateur dont il attend une intervention ou un ordre.

Des informations supplémentaires sont associées à chaque périphérique, ou ce qui revient au même à chaque couple facteur-pilote. Ce sont :

- une zone REPONSE contenant des informations fournies en retour par les instructions AIO ou TIO.
  Dans certains cas cette zone contient un message déposé par le chien de garde ou par le servant.
- l'heure HSIO à laquelle a été déclenchée par SIO le transfert en cours.
  S'il n'y a pas de transfert en cours, HSIO vaut zéro.
- des marques propres à chaque type de périphériques.

3.4.3 Le processus servant

Ce processus assure la conversation entre l'opérateur et le SES.

L'opérateur peut l'appeler par le bouton IT-PUPITRE qui génère une interruption. Lorsque l'appel est accepté, le servant le signale par un indicatif :

!! A VOUS

et attend l'entrée d'un message. Si l'analyse de ce message montre qu'il est correct, le servant exécute l'action demandée : modification d'état des périphériques ou message au pilote. Sinon il sort un message d'erreur, avant de revenir à une position de référence.

Le servant peut être appelé par un autre processus physique ayant un message à sortir. Ce message est déposé dans une boite aux lettres spéciale appelée casier du servant et les processus demandeurs et le servant l'utilisent en producteurs-consommateurs. Les processus producteurs sont :

- les pilotes de périphériques
- le chien de garde
- le processus changemant lorsque le système est oisif
  ("ESOPE EN OMPHALOSKEPSIE")

Dans notre réalisation, l'appel de l'opérateur a priorité sur les messages contenus dans le casier.

```
SERVANT : begin
          P(SCONSSERVANT) ;
          if ITPUP then
               begin
               composition du message !! A VOUS ;
               OUT := 1 ;
               end
          else
               begin // voir commentaire numéro 1 //
               message := tampon (tête) ;
               tête := tête + 1 modulo LCASIER ;
               V(SPRODSERVANT) ;
               composition du message
               end ;

S1 :      SIO ;
          P(SITSERVANT) ;
          if erreur dans SIO then faire couïner le haut-parleur de la 10070 ;
          if OUT = 0 then goto SERVANT ;  // fin de sortie de message //
          if OUT = 1 then
               begin // lecture du message de l'opérateur //
               OUT := 2 ;
               préparer le tampon de lecture ;
               goto S1
               end ;
          // réception du message de l'opérateur //
          ITPUP := false ; OUT := 0 ;
          actions selon message ;
          goto SERVANT ;
          end
```

Les actions du servant en réponse à un message de l'opérateur concernant un périphérique I sont :

```
ERREUR : composition du message !!ERREUR ; goto S1 ;

AVORT :  P(SAVORT(I)) ;
         if  ¬ACTIF(I) then
             begin
             V(SAVORT(I)) ;
             goto ERREUR
             end ;
         if AVORT (I) then
              begin
              V(SAVORT(I)) ;
              goto ERREUR ;
              end ;
         AVORT (I) := true ;
         P(SMUTEX) ; // commentaire numéro 2 //
         if HSIO (I) ¬ = 0 then
              begin
              HIO (I) ;
              HSIO (I) := 0 // commentaire numéro 3 //
              V(SIT) ;
              end ;
         V(SMUTEX)
         if ATTEND (I) then
              begin
              ATTEND (I) := false ;
              V(SOP (I)) ; // on libère le pilote en attente //
              end ;
         V(SAVORT(I)) ;
         goto SERVANT ;

LANCER : if (I¬ = LECTEUR) | ACTIF (I) then goto ERREUR ; // ne concerne que
                                                          le lecteur de cartes //
         AVORT (I) := false ; ACTIF (I) := true ;
         V(SLECTEUR) ;
         goto SERVANT ;

VERROU : if VERROU (I) then goto ERREUR ;
         P(SBLOC (I)) ; // commentaire numéro 4 //
         VERROU (I) := true ;
         goto SERVANT ;
```

```
DEVERROU : if ¬VERROU (I) then goto ERREUR ;
           VERROU (I) := false ;
           V(SBLOC (I)) ; // commentaire numéro 4 //
           goto SERVANT ;

REPRENDRE : if ¬ACTIF (I) | ¬ATTEND (I) then goto ERREUR ;
            ATTEND (I) := false ; REPONSE (I) := 1 ; // message pour le pilote//
            V(SOP (I)) ; // on libère le pilote en attente //
            goto SERVANT ;

CONTINUER : if ¬ACTIF (I) | ¬ATTEND (I) then goto ERREUR ;
            ATTEND (I) := false, REPONSE (I) := 0 ; // message pour le pilote//
            V(SOP (I)) ; // on libère le pilote en attente //
            goto SERVANT ;
```

L'envoi d'une interruption pupitre par l'opérateur a l'effet suivant :

```
if ¬ITPUP then
    begin
    ITPUP := TRUE ;
    V(SCONSSERVANT) ;
    end
```

Commentaires :

1. le servant est seul consommateur dans le casier. Il n'est pas besoin de sémaphore d'exclusion mutuelle.

2. ce sémaphore interdit l'arrivée d'une interruption et la modification de l'état du périphérique pendant la section critique. Comme les interruptions de la 10070 n'utilisent pas les sémaphores, les actions P(mutex) et V(mutex) ne peuvent être obtenues qu'en masquant et en démasquant les interruptions.

3. HSIO(I)¬ = 0 indique qu'un SIO a été lancé par le pilote et que la fin de transfert ne lui a pas encore été notifiée. En le remettant à zéro, et en envoyant un V(SIT), on signale au pilote la fin de transfert simulée. Aucune interruption ne peut plus arriver après un HIO.

4. Pour verrouiller un périphérique, nous utilisons la variable VERROU associée à un sémaphore SBLOC. Nous verrons que SBLOC (I) est utilisé pour bloquer le processus facteur du périphérique (I). Pour coordonner les trois actions : transfert, verrouiller, déverrouiller malgré l'existence de suites d'actions de la forme imprimer, verrouiller , déverrouiller,

verrouiller, déverrouiller,..., verrouiller, demande d'impression, nous avions deux solutions :

a) l'utilisation de SBLOC (I) comme un sémaphore privé initialisé à zéro. Mais il faut alors associer une variable d'état A(I) au facteur et la protection de la variable VERROU(I) par un sémaphore de mutuelle exclusion SMUTEXVERROU ; dans ce cas, à l'aide d'une variable globale supplémentaire, BLOQUE(I), le servant peut se programmer :

```
VERROU : VERROU (I) := true ;
         goto SERVANT ;

DEVERROU : P(SMUTEXVERROU (I)) ;
           if BLOQUE (I) then V(SBLOC (I)) ;
           VERROU (I) := BLOQUE (I) := false ;
            V(SMUTEXVERROU (I)) ;
```

La programmation correspondante du processus facteur est :

```
FACTEUR : P(SFACTEUR (I)) ;
          P(SMUTEXVERROU (I)) ;
          if VERROU (I) then
               begin
               BLOQUE (I) := true
               V(SMUTEXVERROU (I)) ;
               P(SBLOC (I)) ;
               goto FACTEUR ;
               end
          else V(SMUTEXVERROU (I)) ;
```

b) faire systématiquement V(SBLOC (I)) lorsqu'on déverrouille. Pour cela il faut faire l'opération P(SBLOC (I)) à chaque verrouillage. Le facteur se bloque en faisant aussi P(SBLOC (I)). Pour conserver la symétrie, le facteur libéré fait aussi un V(SBLOC (I)).

Nous avons choisi la solution b qui entraine le codage suivant pour les facteurs :

```
lecteur : FACTEUR : P(SFACTEUR (I)) ;
                    P(SBLOC (I)) ;
                    V(SBLOC (I)) ;
                    etc....

imprimante : FACTEUR : P(SFACTEUR (I)) ;
                       if VERROU (I) then
                            begin
                            V(SFACTEUR (I)) ;
                            P(SBLOC (I)) ;
                            V(SBLOC (I)) ;
                            goto FACTEUR ;
                            end ;
                       etc...
```

Ici aussi VERROU est nécessaire pour ne pas immobiliser un message dans la boite aux lettres alors que la seconde imprimante peut le traiter. VERROU n'a pas à être protégé par un sémaphore d'exclusion mutuelle car si le servant le modifie après le test, c'est sans conséquence. En effet, supposons que cette modification arrive :

- si VERROU (I) passe de true à false, le facteur ne se bloquera pas ; il ne fera qu'exécuter une petite boucle inutile,
- si VERROU (I) passe de false à true, le mouvement demandé par le message de la boîte aux lettres se fera en entier, mais le message suivant ne sera pas pris par ce facteur. Cette situation est la même que si la modification de VERROU intervient pendant le mouvement.

3.4.4 Le processus chien de garde

Ce processus lit périodiquement les états des périphériques pour rappeler à l'opérateur que le SES attend un message de sa part ou pour détecter les anomalies de fonctionnement qui ne se signalent pas par des interruptions : état manuel du périphérique, perte d'une interruption.....

```
MEDOR : attendre (délai d'activation périodique) ;
        for I := 1 step 1 until IMAX do // I est le numéro logique de
                                           périphérique //
            begin
            if ¬PILACT (I) & AVORT (I) then goto FIN ;
            if  ATTEND (I) then
                 begin
                 déposer le message !!PERIPH NO I REPONDEZ dans le casier du
                                    servant ;
                 goto FIN ;
                 end ;
            P(SMUTEX) ; // commentaire numéro 1 //
            TIO (I) ;
            if anormal then
                 begin
                 V(SMUTEX) ;
                 déposer dans le casier le message !!PERIPH NO I NON
                                                   OPERATIONNEL ;
                 goto FIN ;
                 end ;
            if  interruption en attente then
                 begin
                 V(SMUTEX) ;
                 goto FIN ;
                 end ;
            if "manuel" then
                 begin
                 HSIO (I) := PENDULE ; // commentaire numéro 2 //
                 V(SMUTEX) ;
                 déposer dans le casier le message !!PERIPH NO I MANUEL ;
                 goto FIN ;
                 end ;
            if HSIO (I) + DELAIMAX <PENDULE then // commentaire numéro 3 //
                 begin
                 HIO (I) ;
                 message dans ETAT (I) ;
                 HSIO (I) := 0 ;
                 V(SSIO (I)) ;
                 end ;
```

```
        V(SMUTEX) ;
FIN :   end
        goto MEDOR ;
```

Commentaires :

1. P(SMUTEX) et V(SMUTEX) sont réalisés par masquage et démasquage des interruptions (voir commentaire du servant).

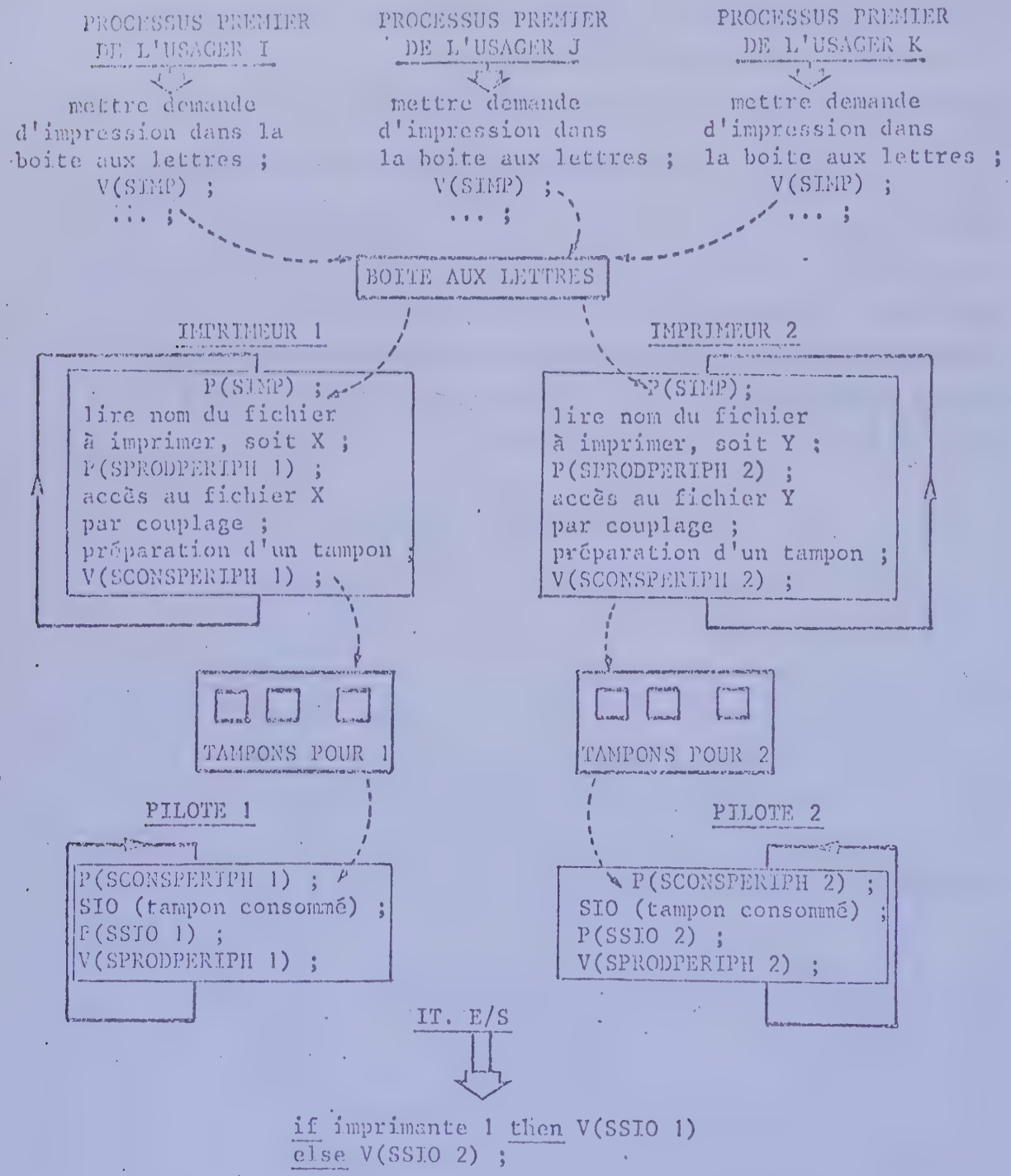
2. PENDULE est une horloge entretenue automatiquement par un circuit câblé.

3. HSIO sert à s'assurer qu'un transfert n'est pas bloqué indéfiniment. Pour cela, HSIO est noté à chaque SIO et remis à zéro en fin d'opération. HSIO est modifié par le chien de garde si le périphérique est en manuel, puisque c'est le seul cas, non classé comme anormal, où le périphérique peut bloquer indéfiniment un transfert. DELAIMAX tient compte de la périodicité du chien de garde, à cause du MANUEL, et de la durée maximum d'un transfert normal. Nous avons pris DELAIMAX égal à 2 fois le délai d'activation du chien de garde.

3.5. Les Imprimantes

3.5.1 Schéma de fonctionnement

Les demandes d'impression de fichier sont déposées dans un fichier spécial, la boite aux lettres. Deux processus facteurs, les imprimeurs, y prennent ces demandes qu'ils traitent en relation avec deux pilotes d'imprimante. Le code des imprimeurs et des pilotes est réentrant.



valeurs initiales des sémaphores : SPRODPERIPH 1 = SPRODPERIPH 2 = N ;
SCONSPERIPH 1 = SCONSPERIPH 2 = 0 ;
SIMP = SSIO 1 = SSIO 2 = 0 ;

3.5.2 Demande d'impression par un processus premier d'usager

La structure de la boîte aux lettres organisée en tampon circulaire est simple.

- le premier mot contient un pointeur de queue PQUEUE
- le second mot, un pointeur de tête PTETE
- les messages sont de taille fixe, TAILLE. Le premier message est à l'emplacement DEBUT et le dernier se termine avant l'emplacement AU DELA.

L'accès à la boîte aux lettres est réglementé par deux sémaphores

- SMUTEXBAL d'exclusion mutuelle, initialisé à 1
- SIMP de consommation initialisé à 0.

Il n'y a pas de sémaphore bloquant la production en cas de boîte à lettres pleine, car nous n'avons pas voulu empêcher un utilisateur conversationnel de faire autre chose. On vérifie à chaque dépôt que la boite n'est pas pleine. Si elle l'est un message de refus est renvoyé à l'utilisateur demandeur.

Dans ce cas le dépôt d'un message par un processus premier s'écrit :

```
P(SMUTEXBAL) ;
ouvrir et coupler la boîte aux lettres
A := PQUEUE + TAILLE ;
if A = AU DELA then A := DEBUT
if A¬ = PTETE then
     begin
     PQUEUE := A ; ERREUR := false ;
     TAMPON (PQUEUE) := message ;
     V(SIMP) ;
     end ;
else ERREUR := true ;
fermer le fichier boîte aux lettres
V(SMUTEXBAL) ;
if ERREUR then envoyer un message à l'utilisateur ;
```

Le retrait de message par les facteurs imprimeurs s'écrit

```
P(SIMP)
P(SMUTEXBAL)
ouvrir et coupler la boîte aux lettres ;
PTETE := PTETE + TAILLE ;
if PTETE = AU DELA then PTETE := DEBUT ;
message := TAMPON (PTETE) ;
fermer la boîte aux lettres ;
V(SMUTEXBAL) ;
```

3.5.3 Facteur de l'imprimante

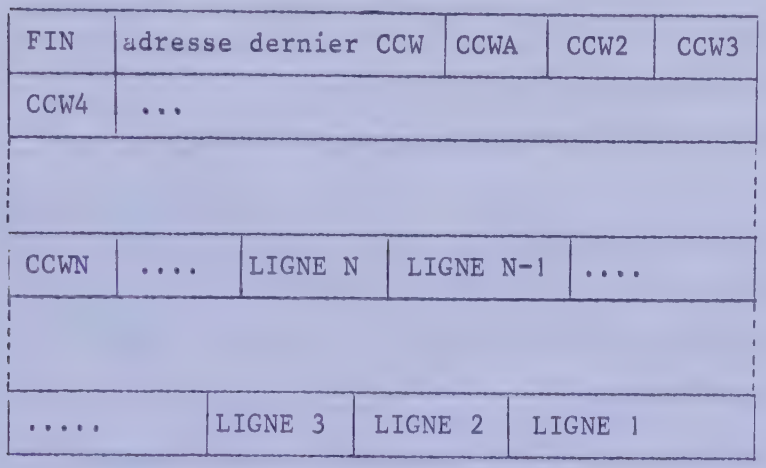
Chaque facteur s'exécute en virtuel chez l'usager PTT et n'imprime que les fichiers de texte

```
IMPRIMEUR : P(SIMP) ;
            if VERROU (I) then
                 begin
                 V(SIMP) ;  // indique que le message n'est pas consom-
                              mé //
                 P(SBLOC(I)) ; // voir le commentaire n° 4 du servant //
                 V(SBLOC(I)) ;
                 goto IMPRIMEUR ;
                 end ;
            ACTIF(I) := true ;
            P(SMUTEXBAL) ;   // SMUTEXBAL protège la boite au lettre //
            prendre un message dans la boite aux lettres ;
            V(SMUTEXBAL) ;
            ouvrir le fichier à imprimer ;
            coupler son premier article ;
            allouer-tampon ;  // colle N pages en mémoire pour servir de
                                 tampons                                 //
            PILACT(I) := true ; // désormais le pilote est actif pour
                                   le chien de garde                     //
            n := m := -1 ;  // indices de tampon courant                 //
         B : P(SPRODPERIPH(I)) ; // sémaphore de production initialisé
                                   à N                                   //
            n := n + 1 modulo N ;
            if AVORT(I) then
                 begin
                 FIN(n) := true ; // commentaire n° 1                     //
                 goto B1
                 end ;
            préparer un tampon ;   // commentaire n° 1                    //
            FIN(n) := if fin-de-fichier then true else false ;
        B1 : V(SCONSOMMPER(I)) ;  // active une phase de consommation     //
            if ¬FIN(n) then goto B ;
            P(SFIN(I)) ;          // commentaire n° 2                     //
```

```
libérer-tampon ;        // restitue les pages collées      //
fermer le fichier à imprimer ;
P(SAVORT(I))  ;
ATTEND(I)) := ACTIF(I) := AVORT(I) := PILACT(I) := false ;
V(SAVORT(I)) ;
goto IMPRIMEUR ;
```

Commentaires :

1. Le tampon n ($0 \leq n \leq N - 1$) contient toute l'information dont le pilote a besoin
   - un booléen FIN pour signaler que ce tampon est le dernier du fichier imprimé
   - l'adresse physique du dernier CCW
   - une suite de CCW formant le programme canal (en adresses physiques)
   - une suite de lignes à imprimer



2. Le dialogue producteur-consommateur se poursuit normalement tant qu'il reste des lignes à imprimer. Quand il n'y en a plus, l'imprimeur attend que le pilote ait fini d'utiliser les derniers tampons avant de détruire ceux-ci. Cette attente est contrôlée par le sémaphore SFIN(I) (initialisé à 0). Les pages collées sont rendues au moniteur, ce qui permet de n'immobiliser de la mémoire que s'il y a effectivement impression.

3.5.4 Pilote de l'imprimante

Chaque pilote s'exécute en physique pour assurer l'impression des tampons que lui communique le facteur. Il se bloque après chaque SIO pour attendre la fin de l'impression demandée. Il signale toute erreur à l'opérateur en mettant un message dans le casier du servant. Lorsqu'il a imprimé un tampon signalé comme dernier par le facteur, il avertit celui-ci par un V(SFIN(I)), et se remet dans son état de référence.

L'entrée-sortie est réalisée en utilisant le chaînage de commandes, ce qui permet d'imprimer plusieurs lignes avec un SIO unique et ce qui exploite le parallélisme entre l'unité centrale et l'unité d'échange. Un tampon contient alors plusieurs commandes CCW dont seule la dernière demande l'envoi d'une interruption de fin de transfert. Celle-ci provoque le réveil du pilote bloqué sur le sémaphore SSIO(I). Par ailleurs, chaque commande demande, en cas d'erreur, l'arrêt immédiat du transfert et l'envoi d'une interruption de fin anormale. Celle-ci provoque le réveil du pilote qui a pour charge d'avertir l'opérateur et de lui demander la conduite à tenir (reprendre l'exécution de la commande erronée, continuer ou annuler le transfert).

En résumé, le pilote se bloque sur un P(SSIO(I)) après avoir lancé le transfert et il n'est activé qu'une fois à l'arrivée d'un des événements suivants :

- fin normale du transfert (interruption de fin de transfert)
- erreur de transfert (interruption de fin anormale)
- annulation du transfert par l'opérateur (commande AVORT).

Il faut être sûr de ne générer qu'un seul V(SSIO(I)) en réponse à un SIO. Pour simplifier la présentation, nous supposerons tout d'abord que le matériel n'envoie jamais qu'une seule interruption en réponse à un SIO ; donc si l'interruption de fin anormale est envoyée, l'interruption de fin de transfert ne le sera pas.

Dans ces conditions le pilote imprimeur s'écrit comme suit :

```
PILIMP : P(SCONSPERIPH(I)) ; // active phase de consommation                    //
         m := m + 1 modulo N ;
         commande := première commande du tampon ;
   L1 : P(SAVORT(I)) ;
         if AVORT(I) then
              begin          // commentaire n° 1                               //
              V(SAVORT(I)) ;
              goto L3
              end ;
         P(SMUTEX) ;
         SIO (Périph n° I, commande) ;
         if "¬OK" then
              begin
              V(SMUTEX) ;    // commentaire n° 2                               //
              ATTEND(I) := true ;
              V(SAVORT(I)) ;
              message-opérateur ; // imprimante non opérationnelle             //
              P(SOP(I)) ;
              goto L1
              end ;
         HSIO(I) := PENDULE ; // commentaire n° 3                              //
         INHIBE(I) := false ; // sauter en première lecture, voir fin du      //
         V(SMUTEX) ;                                          chapitre
         V(SAVORT(I)) ;
         if "manuel" then message-opérateur ;
   L2 : P(SSIO(I)) ;  // commentaire n° 6                                      //
         P(SAVORT(I)) ;
         if AVORT(I) then
              begin
              V(SAVORT(I)) ; // transfert avorté par l'opérateur               //
              goto L3
              end ;

         if fin-anormale then // commentaire n° 4                              //
              begin
              ATTEND(I) := true ;
              V(SAVORT(I)) ;
              message-opérateur ; // imprimante en erreur                      //
              P(SOP(I)) ;
              if AVORT(I) then goto L3 ;
              if REPONSE(I) = "reprendre" then                                //
                   begin   // reprise de l'entrée-sortie                      //
                   TIO (Périph. n° I) ;
                   commande := commande-en-cours ;
                   end
              else
                   begin     // on ignore l'erreur et on continue            //
                   TIO (Périph. n° I) ;
                   commande := commande-en-cours + 1 ;
                   end ;
              goto L1 ;
              end ;
         // on traite ici le cas de la fin de transfert des 1 lignes          //
         V(SAVORT(I)) ;

   L3 :  if FIN(m) then V(SFIN(I)) ; // commentaire n° 5                       //
         V(SPRODPERIPH(I)) ; // active phase de production du facteur         //
         goto  PILIMP ;
```

Commentaires :

1. Si l'opérateur a avorté la sortie du fichier, alors l'opération d'entrée-sortie est sautée et le pilote consomme "à vide" les tampons pleins qui lui ont éventuellement été fournis par le facteur ; il se repositionne ensuite à son état de référence derrière le sémaphore SCONSPERIPH(I).

2. L'imprimante se trouve à l'état non opérationnel. Un message est passé à l'opérateur qui pourra ou relancer le transfert ou l'avorter.

3. Le transfert a été accepté. HSIO(I) est remis à l'heure pour permettre au processus chien de garde de contrôler périodiquement si l'interruption de fin de transfert n'a pas été perdue. Lorsque le chien de garde détecte le mode manuel il remet HSIO(I) à l'heure.

4. Une erreur s'est produite, au cours de l'impression d'une ligne entraînant du même coup l'arrêt de l'imprimante. Le pilote signale l'erreur à l'opérateur qui peut :
   - soit avorter la sortie du fichier
   - soit continuer à partir du point où en était resté le transfert
   - soit reprendre la ligne défectueuse en cours d'impression.

5. Dans le cas où le tampon traité est le dernier, le pilote réveille le facteur par la primitive V(SFIN(I)).

Compléments :

1. En réalité, l'imprimante ne fonctionne pas tout à fait comme nous l'avons supposé jusqu'à présent. Les deux interruptions - fin de transfert et fin anormale - peuvent arriver toutes les deux pendant l'exécution d'une commande et ce à des instants différents. Ceci se passe lorsque l'opérateur appuie sur le bouton "remise à zéro" de l'imprimante, alors qu'une impression s'exécute. Pour en tenir compte et inhiber la seconde interruption, nous avons introduit la variable booléenne INHIBE(I) inscrite à false par le pilote dès que le SIO est accepté et remise à true à l'arrivée de la première interruption.

Le programme d'interruption associé au pilote imprimeur s'écrit :

```
P(SMUTEX) ;
AIO ;
HSIO(I) := 0 ;
if ¬INHIBE(I) then
     begin
     INHIBE(I) := true ;
     V(SSIO(I)) ;
     end ;
V(SMUTEX) ;
```

Si l'interruption de fin de transfert arrive assez longtemps avant l'interruption de fin anormale, il se peut qu'un nouveau SIO soit émis par le pilote. L'opération sera refusée et un message envoyé à l'opérateur.

2. La même méthode doit être employée chaque fois qu'une interruption est simulée par un V(SSIO(I)). Ceci se produit :
   - dans le chien de garde, lorsqu'il a détecté la perte d'une interruption,
   - dans le servant, en exécutant la commande d'AVORT.

Remarque :

Sur le matériel mis à notre disposition, l'action sur le poussoir MANUEL a les effets suivants :

- si une commande (CCW) est en cours, elle est terminée dans tous les cas, ce qui entraine l'envoi éventuel d'une interruption de fin de transfert
- l'indicateur "manuel" est mis à 1.

Il arrive fréquemment que cette mise à 1 ne soit pas encore faite lorsque l'interruption de fin de transfert est envoyée. Un TIO consécutif à celle-ci ne lira qu'un zéro. Dans ce cas le passage en manuel ne peut être détecté que par le chien de garde.

Comme le matériel doit être modifié pour que le signal manuel arrive à temps (ce serait mineur : il suffirait d'ôter une temporisation), nous avons conçu le pilote imprimeur pour qu'il détecte le passage en manuel. Il faut pour cela :

- générer une interruption de fin de transfert pour chaque ligne,
- faire réveiller le pilote par le programme d'interruption lorsque le mode "manuel" est présent,
- faire déposer un message au servant par le pilote. Ce message ne peut être déposé directement par le programme d'entrée-sortie car il faut pouvoir bloquer le producteur lorsque le casier est plein.

3.6 Le lecteur de cartes

3.6.1 Schéma général

Deux processus coopèrent à la gestion du lecteur de cartes : le producteur est le pilote attaché au lecteur, processus physique et le consommateur est le facteur du lecteur, processus en virtuel.

L'opérateur déclenche la lecture des cartes par un ordre au servant et celui-ci active le facteur qui, avant de s'engager avec le pilote dans le dialogue production-consommation, demande au moniteur l'allocation des pages

collées utilisées comme tampon.

La lecture cesse soit lorsque le pilote rencontre une carte !FIN, soit lorsque l'opérateur décide d'arrêter le transfert par un ordre AVORT donné au servant.

3.6.2 Pilote du lecteur

Le pilote, pour arrêter la lecture sur une carte !FIN, doit être activé après chaque lecture de carte. Pour cette raison, le chaînage de commande n'a pas été utilisé. A chaque fin de transfert d'une carte, qu'il soit correct ou non, une interruption est générée et le pilote est appelé.

Le pilote est décrit par l'algorithme suivant :

```
PILLECT  : P(SPILLECT) ;
           PILACT(I) := true ;        // Commentaire n° 1                  //
           n := 0 ;                   // n est l'indice du tampon courant  //
TAMPON   : P(SPRODPERIPH(I)) ;        // Sémaphore de production, initia-
                                         lisé à N                          //
           FIN(n) := false ; NOMBRE(n) := 0 ; // Commentaire n° 2           //
           1 := 0 ;                   // l est l'indice de la ligne
                                         courante                          //
LIGNE    : P(SAVORT(I)) ;             // Commentaire n° 3                  //
           if AVORT (I)   then
              begin                   // Transfert avorté par l'opérateur  //
              V(SAVORT(I)) ;
              goto   FIN ;
              end ;
           P(SMUTEX) ;                // Commentaire n° 4                  //
           SIO(Periph n° I, LIGNE(l)) ;
           if  "¬OK"   then
               begin                  // Le lecteur n'est pas en état de
                                         marche                            //
               V(SMUTEX) ;
               ATTEND(I) := true ;
               V(SAVORT(I)) ;         // Commentaire n° 3                  //
               message à l'opérateur ;
               P(SOP(I)) ;            // Le pilote attend l'opérateur      //
               goto   LIGNE ;
               end ;
           HSIO(I) := PENDULE ;
           V(SMUTEX) ; V(SAVORT(I)) ; // Commentaire n° 3 et 4             //
           if  "manuel" then message à l'opérateur ;
           P(SSIO(I)) ;                 // Commentaire n° 5                  //
```

```
            P(SAVORT(I)) ;
            if AVORT(I) then
                begin
                V(SAVORT(I)) ;
                goto   FIN ;
                end ;
            if "erreur de transmission"  then
                begin
                ATTEND(I) := true ;
                V(SAVORT(I)) ;
                message à l'opérateur ;
                P(SOP(I)) ;
                if AVORT(I) then goto FIN ;
                if REPONSE(I) = "reprendre" then goto LIGNE ;
                end
            else V(SAVORT(I)) ;
            if   LIGNE(l) = "carte !FIN" then goto FIN ;

            NOMBRE(n) := NOMBRE(n) + 1 ;
            if  "tampon plein" then
                begin
                V(SCONSPERIPH(I)) ; // Sémaphore de consommation          //
                n := n + 1 modulo N ;
                goto TAMPON ;
                end ;
            l := l + 1 ; goto LIGNE ;
FIN :       PILACT(I) := false ;  // Commentaire n° 1                      //
            FIN(n) := true ;
            V(SCONSPERIPH(I)) ;
            goto PILLECT              // Retour à la position de référence  //
```

Commentaires:

1. Le pilote se place sous la surveillance du chien de garde dès qu'il est lancé et s'en retire dès que la dernière carte est lue (PILACT est une variable globale).

2. Chaque tampon m ($0 \leq m \leq N-1$), fourni par le pilote au facteur, contient en plus des images des cartes lues :

- Un indicateur, FIN(m), mis à "vrai" lorsque le tampon correspondant est le dernier (le dialogue producteur-consommateur est alors terminé),
- Un compteur, NOMBRE(m), indiquant le nombre de cartes lues dont l'image se trouve dans le tampon correspondant.

3. Cette section critique est nécessaire pour que le servant ne puisse pas avorter le transfert entre le test de l'AVORT par le pilote et le lancement du SIO. D'autre part, dans le cas d'un refus du SIO l'AVORT ne doit pas être positionné avant que l'indicateur ATTEND ne le soit (le pilote resterait bloqué derrière SOP(I)).

4. Cette section critique est réalisée par masquage contre toutes les interruptions, et a pour but de rendre indivisible le lancement du SIO et la mise à l'heure de HSIO(I) ; si l'interruption de fin de transfert arrivait, en effet, entre ces deux actions, sans que la séquence d'instructions ne soit masquée, le contenu de HSIO(I) ne serait pas "homogène" (voir le commentaire n° 5).

5. Traitement d'une interruption en provenance du lecteur :

```
  IT d'E/S
pour le lecteur ——→ P(SMUTEX) ;
                    AIO ;
                    REPONSE(I) := "indicateurs fournis par le AIO" ;
                    HSIO(I) := 0 ;
                    V(SSIO(I)) ;
                    V(SMUTEX) ;
```

3.6.3 Facteur du lecteur

Il est activé chaque fois que le pilote a rempli un tampon d'une page et est chargé de gérer les fichiers lus en créant les fichiers internes correspondants et en les cataloguant. Pour cela, tout paquet de cartes introduit au lecteur doit être précédé d'une carte de commande indiquant le nom du fichier, le nom du client propriétaire et le format (EBCDIC ou binaire). Une option particulière permet la transmission du fichier au travail de fond.

Pour limiter les conséquences d'erreurs de manipulation dues à l'opérateur, nous avons décidé que le lecteur ne détruirait jamais de fichier et ne pourrait qu'en créer de nouveaux.

Le facteur est décrit par l'algorithme suivant :

```
LECTEUR : P(SLECTEUR) ;              // commentaire n° 1 //
          P(SBLOC(I)) ;              // commentaire n° 2 //
          V(SBLOC(I)) ;
          allocation des pages-collées ;
          V(SPILLECT) ;              // lancement du pilote //
          ETAT := 0 ;
```

```
          n := 0 ;      // variable locale à ce processus      //
TAMPON :  P(SCONSPERIPH(I)) ; // Sémaphore de consom-
                              mation, initialisé à 0      //
          l := 0 ;
          for  i := NOMBRE(n) step - 1 until 1 do
               begin
               if AVORT(I) then
                    begin
                    if ETAT = 1 then RAZ-du-fichier-en-cours ;
                    while ¬FIN(n) do
                         begin // Consommation de tous les   //
                         V(SPRODPERIPH(I)) ; // tampons, sauf
                                               le dernier   //
                         n := n + 1 modulo N ;
                         P(SCONSPERIPH(I)) ;
                         end ;
                    ETAT := 2 ;
                    goto FIN ;
                    end ;
               if LIGNE(1) = "Carte de commande" then
                    begin
                    if ETAT = 1 then catalogage -du-fichier-en-
                                                 cours
                                else ETAT := 1 ;
                    analyse ;
                    if "erreur" then
                       begin
                       message-à-l'opérateur ;
                       ETAT := 2 ;
                       end
                    else création-d'un-fichier-de-travail ;
                    end
               else
               if ETAT = 1 then rangement (LIGNE(1))
               else
               if ETAT ¬ = 2 then
                    begin    // Il n'y a pas de carte de
                                commande                    //
                    message-à-l'opérateur ;
                    ETAT := 2 ;
                    end ;
               l := 1 + 1 ;
               end ;
          // En sortant de la boucle for toutes les lignes du
             tampon courant ont été traitées  //
          if  ¬FIN(n)   then
               begin     // Ce n'est pas le dernier tampon    //
               V(SPRODPERIPH(I)) ;
               n := n + 1 modulo N ;
               goto TAMPON ;
               end ;
FIN    :  V(SPRODPERIPH(I)) ;   // Libération du dernier tampon//
          if ETAT = 1 then catalogage -du-fichier-en-cours ;
          Libération-des-pages-collées ;
          P(SAVORT(I)) ;
          ACTIF(I) := AVORT(I) := false ;
          V(SAVORT(I)) ;
          goto LECTEUR ;   // Retour à la position de référence//
```

Commentaires :

1. Dans ce schéma ETAT est une variable entière qui a pour significations :

0 - La ligne courante doit être une carte de commande.
1 - Il y a un fichier créé : la ligne courante doit y être rangée si ce n'est pas une carte de commande.
2 - Il n'y a pas de fichier créé (carte de commande erronée ou absente) : la ligne courante n'est pas rangée tant que ce n'est pas une carte de commande.

2. La variable VERROU(I) est utilisée pour contrôler le verrouillage du périphérique I et le sémaphore SBLOC(I) (initialisé à 1) pour bloquer le facteur associé à ce périphérique lorsque le verrouillage a été demandé par l'opérateur au servant. Les actions effectuées par le servant sont les suivantes :

```
VERROU : if VERROU(I) then "erreur" ;
         P(SBLOC(I)) ;
         VERROU(I) := true ;
```

```
DEVERROU : if ¬VERROU(I) then
           "erreur" ;
           VERROU(I) := false ;
           V(SBLOC(I)) ;
```

Se reporter aussi au commentaire n° 4 du processus servant (§ 3.4.3).

3.7 Performances du SES avec lecteur de cartes et imprimantes :

Les performances du SES d'ESOPE peuvent être évaluées sous l'angle des entrées-sorties seules ou en fonction de l'augmentation du temps de réponse aux utilisateurs conversationnels.

Les résultats ci-dessous, comparant ESOPE à d'autres systèmes d'exploitation du CII 10070, prouvent largement l'efficacité des mécanismes généraux employés :

	BPM vide	BPM chargé	SIRIS 7 vide	ESOPE vide	ESOPE chargé
Entrée d'un fichier de 2800 cartes	2mn 24s + 1mn	2mn 30s + 1 mn	2mn 18 s + ?	2 mn 16s	2 mn 16 s
Impression du même fichier	5mn 11s		4mn 09 s	4 mn 02 s	

Nota : Les temps indiqués pour BPM et SIRIS 7 sont décomposés en temps d'entrée des cartes dans la zone des symbionts (2 mn environ) et en temps de transfert du symbiont vers le fichier (1 mn environ). ESOPE n'a pas besoin de zone symbiont intermédiaire.

Le temps de réponse du système aux utilisateurs conversationnels, mesuré assez grossièrement avec une charge moyenne (de 4 à 10 usagers), augmente sensiblement (de 1,1 s à 1,4 s en moyenne) lorsque le SES fonctionne sans arrêt.

L'étude du temps de réponse sera reprise avec un simulateur de charge fournissant des conditions d'expérience reproductibles. Nous essaierons de l'améliorer par les modifications suivantes :

- changement de la priorité de l'usager SES (actuellement le plus prioritaire).

- limitation du nombre de pages physiques occupées par le SFS : les pages libérées par un usager sont rendues au système (listes des pages libres ou quasi libres) qui les chaîne en queue de liste pour faciliter la récupération en mémoire centrale des pages fréquemment utilisées ([3]). Comme la probabilité de récupération des pages du SES est négligeable, on peut au contraire les mettre en tête de liste, ce qui entrainera leur réutilisation plus rapide, par le SES éventuellement, et limitera le nombre de pages physiques occupées par des fichiers en entrée ou en sortie.

3.8 Bibliographie

[1] BAUDET, FERRIE, KAISER, MOSSIERE. Entrées-sorties dans un système à mémoire virtuelle. Congrès AFCET, Grenoble, nov. 1972

[2] BAUDET, FERRIE, KAISER, MOSSIERE. Entrées-sorties d'ESOPE et usager PTT note IRIA/ESOPE/A/21 du 29.05.72

[3] CII 10070 - Manuel de référence

[4] HABERMANN. Synchronization of communicating processes. CACM, mars 1972.

CONCLUSION

Courbe la tête, jier Sicambre,
adore ce que tu as brûlé,
brûle ce que tu as adoré.

St Rémi

Nous allons revenir sur les concepts introduits dans la seconde partie et essayer de les évaluer à la lumière des réalisations décrites dans la dernière partie et de l'expérience que nous en avons eue pendant la construction et la mise au point du système ESOPE. Nous nous attacherons à montrer les aspects positifs des solutions choisies, mais aussi leurs limites et les voies qui sont ainsi ouvertes à de nouvelles recherches. Notre analyse portera successivement sur la coopération des processus, l'accès et la protection de l'information, l'allocation des ressources, et enfin la méthodologie de réalisation du système.

D'une manière générale, le respect du précepte cartésien de n'utiliser qu'un petit ensemble de notions de base et de n'augmenter leur nombre que par incapacité à résoudre les problèmes avec les notions déjà existantes, s'est révélé très efficace. Cela a permis d'éviter que plusieurs termes ne désignent la même chose et a aboutit à former une unité de vocabulaire, de pensée et de raisonnement qui a grandement facilité la communication des idées entre les membres du projet et la compréhension de l'ensemble du système par chacun d'eux. A chacune de ces notions nous avons alors fait correspondre un mécanisme et un seul lors de la réalisation du système, ce qui a simplifié la phase de mise au point.

Coopération des processus

Lors de la décomposition du système en activités parallèles, nous avons introduit les notions de processus et de sémaphore pour pouvoir agir sur un programme en cours d'exécution et pour régler la coopération des processus. On a pu constater dans le chapitre trois l'ampleur de leur utilisation. Nous n'insisterons donc pas sur l'utilité de leur emploi qui nous semble acquise, mais nous nous arrêterons sur quelques remarques qu'il engendre.

a) Nous avons pris une définition volontairement vague du processus. Ceci n'a pas d'importance dans le cas précis d'ESOPE, car compte tenu du contexte de la machine utilisée et des structures de données y décrivant les processus, il n'y a jamais eu d'ambiguité. Comme conséquence notre définition ne correspond pas en toute rigueur à celle qui est employée, de manière aussi vague et avec d'autres contextes, dans d'autres systèmes. Divers auteurs se sont employés [1, 2, 3], sans faire encore l'unanimité, à préciser cette notion de façon à la dégager des particularités de chaque réalisation.

b) Les sémaphores sont un outil bien adapté pour traiter les problèmes de synchronisation entre processus et une excellente aide au raisonnement sur des activités parallèles, à condition d'adopter une approche unique des problèmes et de rejeter, comme nous l'avons fait, les notions d'interruption et de masquage de programme. Par contre, la manipulation des sémaphores avec P et V n'impose aucune structure élaborée de donnée, de programme ou de système chez leur utilisateur. Ces primitives sont comparables aux instructions câblées de la machine. Tous les problèmes habituels de structuration de l'information demeurent : protection contre les actions interdites, création d'opérations nouvelles.

Le problème se complique dans ce dernier cas, si on veut que la procédure définisse une nouvelle opération atomique d'un niveau plus élevé, car cette atomicité, ou indivisibilité, doit être vraie aussi, du point de vue des autres processus. Cela conduit à envisager non seulement de nouvelles procédures, mais de nouvelles "primitives" dont l'atomicité doit être assurée par le superviseur. Partant de la définition historique du sémaphore, considérons les extensions que nous avons été amenés à y faire et celles dont nous préconisons l'étude.

b1) Nous avons tout d'abord réglementé et protégé l'emploi des sémaphores et des processus. Dans ESOPE, la portée des noms est contrôlée et limitée à l'usager qui les crée. Ainsi le nom d'un processus, le nom d'un sémaphore ne sont accessibles qu'aux processus d'un même usager. Nous assurons ainsi la protection des usagers entre eux. Les sémaphores de moniteur ne sont accessibles à aucun usager. La communication des usagers avec le moniteur se fait à leur insu dans des procédures du

système appelées par déroutement (par exemple, trappe pour défaut de page) ou par instruction CAL. Le sémaphore lui-même, variable entière et file d'attente associées, n'est accessible à aucun processus car les opérations indivisibles P et V se font seulement par l'intermédiaire d'instructions CAL qui seules ouvrent l'accès à des données et des procédures écrites et protégées dans la mémoire physique.

b2) Après les opérations isolées, ce sont les suites d'opérations, qu'il faut contrôler si celles-ci peuvent mettre en échec le fonctionnement du système. C'est le cas si un mauvais emploi des sémaphores conduit au blocage des processus, par étreinte fatale. C'est aussi le cas si un processus s'arrête ou est détruit dans une section critique, par suite d'une erreur de programmation ou d'une panne, interdisant tout autre accès en section critique. En limitant la portée des noms à l'usager, nous avons de ce fait limité l'étreinte fatale et la propagation des erreurs aux processus d'un usager. Quant aux processus de système, ils sont écrits de façon que l'entrée dans les sections critiques se fasse, toujours dans le même ordre, ce qui est une condition suffisante pour éviter l'étreinte fatale.

b3) Nous avons aussi cherché à augmenter la puissance des opérations sur les sémaphores, mais, ceci n'est intéressant que si l'indivisibilité des opérations est assurée par le moniteur. Le paragraphe 2.2.10 du chapitre deux le montre très simplement. La primitive D(sem) a été introduite pour un besoin spécifique, celui de l'expression d'événement impulsionnel. A l'usage, nous y faisons deux objections. Tout d'abord elle est, comme les P et V, à un niveau trop bas, trop proche des instructions câblées. Mais surtout elle ne répond pas aux conditions mises en avant à ce jour pour démontrer la validité des programmes, car elle fait intervenir les vitesses d'exécution des processus : si, toutes choses égales par ailleurs, les vitesses relatives des processus changent, la présence de primitives D entraine que les résultats obtenus peuvent changer. Une autre approche [4, 5] a été essayée depuis le début de notre travail : elle règle la coopération entre processus à l'aide de messages gérés par le système. Cette extension eut certainement été très utile dans ESOPE, en uniformisant les méthodes de communication entre processus. Nous ne l'avons pas ajoutée, car en fait elle n'a d'intérêt que pour la communication entre usagers. Des recherches plus théoriques sont en cours pour dégager des procédures indivisibles d'emploi général [6, 7, 8]. La composition d'événements et la prise en compte de délais ne se font pas simplement avec les

sémaphores. Nous avons assimilé la fin d'un délai à une opération V en définissant un sémaphore du moniteur, S:CHRONOS et une primitive suspendre (délai). L'effet de celle-ci est de mettre le processus demandeur dans la file associée à S:CHRONOS. Cette file est ordonnée selon les délais croissants. La valeur du premier délai sert à charger un compte-temps dont le passage à zéro active une primitive V (S:CHRONOS). Il reste à trouver une opération générale de composition de sémaphores analogue à ce qui existe dans le langage PL1, pour les événements.

L'intérêt de ces extensions est grand car en dehors des simplifications qu'elles peuvent introduire dans la réalisation des systèmes, elles permettront d'obtenir plus facilement des preuves de validité des algorithmes mettant en jeu le parallélisme.

b4) L'utilité des sémaphores peut encore s'analyser en fonction du degré de parallélisme que l'on veut donner au système d'exploitation. Si on construit un système entièrement centralisé [9], où toute action impliquant l'utilisation de ressources commune fait appel à un moniteur central, où l'activité de celui-ci entraine l'arrêt de toute autre activité pouvant utiliser ou modifier les mêmes ressources que lui et où la communication entre les processus et le moniteur central se fait par des variables locales à chaque processus, alors le sémaphore est inutile. C'est ainsi que le système Haliotis a été construit. Par contre, dès que le système d'exploitation est tant soit peu distribué, ce mécanisme, ou un mécanisme analogue, trouve sa place. Esope comprend une partie centralisée, le moniteur s'exécutant en adresses physiques, décrit et réalisé sous forme de processus parallèles plus prioritaires que les processus des usagers. Toute cette partie aurait pu être écrite, sans sémaphore, comme une seule activité cyclique consommant les messages des usagers. Nous ne l'avons pas fait pour conserver partout les mêmes concepts et parce que ceux-ci nous sont apparus utiles pour analyser les actions qui dans ce moniteur pourraient être faites en parallèle. Une analyse plus approfondie et des mesures sur le système actuel devraient permettre d'évaluer l'utilité d'une décentralisation de cette partie dans le cas d'un multiprocesseur. Par ailleurs, le processus premier de chaque usager utilise une procédure réentrante en adresses virtuelles. Tous les processus premiers agissent en parallèle car il est rare que deux d'entre eux demandent les mêmes ressources à un instant donné.

Accès à l'information

a) C'est pour l'accès à l'information que la simulation de mécanismes non câblés est le plus coûteux, aussi avons-nous employé au maximum les dispositifs existants, parfois au prix d'un manque de souplesse. La technique du couplage nous a permis d'utiliser le mécanisme de pagination au delà de sa destination première, l'allocation de la mémoire principale, et d'en faire une méthode d'adressage dans laquelle la mémoire virtuelle n'est plus qu'un ensemble d'adresses et non un support d'information. La mémoire topographique, n'est plus que la description de l'application qui à un instant donné fait passer de l'adresse au support d'information. Il en résulte, en laissant au programmeur la possibilité de définir cette application, que l'information manipulée par le programme n'est plus limitée par la taille de la mémoire virtuelle. Cette technique s'est révélée très utile et a permis d'étendre facilement l'emploi d'ESOPE. En effet nous avons construit un système de fichiers très rudimentaire mais grâce au couplage, des programmes importants en taille ou en nombre de fichiers, un interpréteur du langage PL1 et un système de gestion de données graphiques, ont pu être installés sans trop de problèmes.

Deux modifications de la machine apporteraient une amélioration immédiate à ESOPE. Que le déroutement pour défaut de page fournisse le numéro de page virtuelle et non l'instruction qui en est l'origine. Que les programmes soient translatables en mémoire virtuelle. A plus long terme, nous préconisons l'étude d'une segmentation de la mémoire virtuelle, rattachée à la notion d'usager.

b) C'est dans l'accès à l'information qu'interviennent aussi les problèmes de protection du système. Ici encore nous avons dû faire avec ce que nous a donné le matériel câblé : protections attachées à la mémoire virtuelle, à la mémoire physique et appels au superviseur. L'usager est encore intervenu, comme possesseur de certaines protections et de certains droits. Toutefois, les processus d'un usager ont à être protégés les uns contre les autres. Nous ne sommes parvenus qu'à protéger le processus premier, en utilisant les clés d'écriture à la mémoire physique. Il a été plus facile de contrôler les emplois des primitives lors des appels au superviseur, car nous avons là un point d'entrée unique où le superviseur peut faire toutes les vérifications nécessaires. Nous

avons associé à chaque usager d'une part et à chaque processus d'autre part, une liste des actions, ou des classes d'actions, qu'il peut entreprendre. Cette méthode se généraliserait sans peine si les accès de chaque processus à la mémoire virtuelle étaient aussi contrôlés par un système câblé de clé figurant dans le mot d'état programme et de verrou installé dans la mémoire topographique de l'usager.

Allocation des ressources

L'allocation de ressources repose sur la notion d'usager. Un travail complémentaire au nôtre [10] fait l'analyse et l'évaluation des méthodes d'allocations employées. Nous nous limiterons ici à considérer l'emploi de l'usager comme allocataire des ressources. Nous avons été très génés lors de la construction du système parce que les mécanismes câblés de la machine sont destinés à un moniteur bâti sur l'association d'un processus à une mémoire virtuelle. Tous les mécanismes que l'on souhaiterait trouver associés au processus sont en fait associés à la mémoire virtuelle : réentrance des programmes, réentrance des interruptions et des déroutements, protection, toutes informations propres au processus et gardées entre deux activités de celui-ci. Nous avons dû pallier cette association prématurée par une gestion compliquée de la mémoire et quelquefois par l'arrêt momentané du parallélisme des actions à l'intérieur d'un usager. En revanche, le fait de ne changer de topographie mémoire qu'à chaque changement d'usager et non de processus représente un gain de place et de temps. Nous avons conservé la notion d'usager qui nous a été très utile pour traduire la séparation d'activités indépendantes et associer des droits particuliers à chacune d'elles, par exemple : protection, ressources allouables, processus premier propre à chaque classe d'usager. On a vu dans le chapitre trois l'extension prise par cette notion.

Méthodologie

La méthodologie de construction que nous avons suivie pour ESOPE est marquée par la taille de notre équipe. Les analyses et les décisions importantes ont pu être prises en commun, ce qui simplifiait les problèmes d'organisation. Nous restreindrons notre analyse aux techniques que nous avons utilisées.

a) L'emploi d'un langage d'écriture plus élaboré qu'un assembleur nous parait une condition essentielle de notre réussite. Un système d'exploitation est un programme suffisamment complexe et il est indispensable d'écarter tous les problèmes qui distraient de l'essentiel. Le LP 10070 s'est révélé très utile en permettant l'expression claire de la structure des procédures et par là a apporté la meilleure documentation du système. Les reproches que nous lui adressons traduisent tous le fait que c'est un langage de trop bas niveau. Ainsi, il faudrait lui adjoindre :

- la gestion automatique des registres de travail, tout en gardant la possibilité de se réserver des registres particuliers,
- des types d'information plus élaborés et en particulier des structures incorporant des pointeurs,
- des dispositifs de protection de l'information comme la déclaration dans l'appel d'une procédure des variables globales qu'elle peut modifier ou comme n'autoriser l'accès à des variables que par l'intermédiaire de procédures ayant ce pouvoir. Cette dernière exigence peut être contrôlée en introduisant l'emploi du couplage dans les appels et les retours de procédure et là seulement.

b) Les étapes de notre construction et les essais associés à chaque étape ont exprimé notre méthodologie : faire apparaître très tôt les principaux processus du système, quitte à les faire s'exécuter dans des conditions simplifiées.

La réalisation du système s'est faite en plusieurs étapes :

- construction du mitan, ensemble des primitives du moniteur assurant la gestion des processus et des sémaphores et mise en place des processus du moniteur et des processus premiers de plusieurs usagers.
  Dès cette étape, nous avons introduit autant d'usagers conversationnels que de consoles disponibles (2 d'abord, puis 5, 16 et 24) et nous leur avons permis d'utiliser le système avec des programmes simples : un éditeur rudimentaire, une machine de bureau, un programme en boucle de longue durée. Dans cette étape, nous n'avons introduit que la gestion de l'unité centrale.

Les problèmes liés au parallélisme, ceux dont la mise au point ne peut être faite complètement qu'en faisant fonctionner le système, se manifestent alors avant que le système ne soit trop gros et leur mise au point en plus facile. En fait, cette mise au point s'est réalisée très vite avec l'emploi d'un dispositif simple qui s'est montré très utile : comme l'allocation de l'unité centrale n'est modifiée que par l'action des primitives P, V, D, "créer un processus", "détruire un processus" - la fin de quantum génère aussi un V - et que cela n'apparaît que dans une seule procédure qui leur est commune, nous avons associé à celle-ci un tampon circulant dans lequel sont notés, à chaque changement d'allocation, le nom du dernier processus, le temps pendant lequel il a eu l'unité centrale et l'adresse de l'instruction où il s'est arrêté.

- gestion de la mémoire physique par les processus du moniteur. Cette étape s'est faite en deux coups. Tout d'abord l'ensemble des programmes des usagers tenait en mémoire et la topographie n'a servi qu'à allouer des pages physiques quelconques. Ensuite l'emploi du disque a été introduit, avec des fichiers définis à l'écriture du système.
- gestion de la mémoire virtuelle de l'usager. A ce point nous avons construit le mécanisme du couplage, puis le système des fichiers internes.
- communication avec l'utilisateur du système. La procédure du processus premier comporte désormais une partie servant à l'entrée d'un client et une partie dépendant du type d'usager qu'il faut créer :

  - usager conversationnel normal
  - usager temps réel
  - usager visu
  - usager PTT (dans ce cas, nous avons aussi introduit des processus facteurs)
  - usager fond

Cette étape correspond au développement des services que le système offre à l'utilisateur et au créateur de traducteur.

Amorcée comme nous l'avons dit plus haut dès la première étape pour pouvoir faire des essais significatifs, elle n'a pu se terminer que lorsque tous les mécanismes de base ont été réalisés.

- mise en place des traducteurs. Cette phase correspond à l'intégration sous système des travaux d'autres équipes de l'IRIA qui ont équipé ESOPE d'un éditeur de texte, d'un interpréteur conversationnel de PL/1, d'un compilateur LP 10070 et d'un relieur de modules compilés séparément.

Cette méthode nous a permis, associée à l'emploi du LP 10070, de construire un système dont la mise au point ne fut pas un problème compliqué. Il est vrai que grâce aux concepts de processus et de sémaphores, nous avons pu analyser entièrement les problèmes les plus difficiles, avant de les programmer.

Et si c'était à faire ? Comment ferions-nous ? Quels enseignements avons nous dégagés de notre expérience présente ? Nous avons été amenés à prendre des décisions à chaque étape de construction et chacune de ces décisions nous a lié de plus en plus. Nous ferions porter en premier notre effort sur les points dont les conséquences nous ont paru les plus étendues. Ainsi nous traiterions les problèmes dans l'ordre suivant :

- définition d'une méthodologie
- étude de l'adressage et des fichiers
- gestion du parallélisme
- allocation des ressources. En dernier, parce que le plus facile à modifier.

Enfin il faudrait restreindre l'ampleur du système et définir un projet beaucoup moins ambitieux de façon à approfondir davantage chacune des parties.

En conclusion notre travail a permis de montrer que la conception et la construction de systèmes informatiques, activités qui nécessitent l'étude de problèmes fondamentaux comme le parallélisme des actions, la gestion de l'information et l'allocation des ressources, peuvent être réalisés à l'aide d'un petit nombre de concepts et qu'en particulier les notions de processus et de sémaphore conviennent pour le traitement du parallélisme.

Bibliographie

[1] DENNIS. Concurrency. Advanced course on software engineering, Munich 1972.

[2] HABERMANN. On the harmonious cooperation of abstract machines. Thèse. Eindhoven 1968.

[3] COFFMAN, DENNING. Computer operating systems : modeling and analysis. Livre à paraître.

[4] HANSEN. The nucleus of a multiprogramming system CACM 13,4.

[5] DETLEFSEN, MORRIS. A virtual processor for real time operation. Software engineering V.1 (TOU).

[6] HOARE. Towards a theory of parallel programming. International seminar Belfast 1971.

[7] HANSEN. A comparaison of two synchronizinz concepts. Acta Informatica 1,3 (1972).

[8] VANTILBORGH, VAN LAMSWEERDE. On an extension of Dijkstra's semaphore primitive. MBLE report R 192 (1972).

[9] GIRAULT. Un des systèmes de multiprogrammation réalisés à 1'Institut de programmation. RIRO B-2- 1971.

[10] KRAKOWIAK. Conception et réalisation de système à accès multiples : allocation de ressources. Thèse à paraître.

ANNEXE 1.

ANALYSE DE QUELQUES ERREURS DE PROGRAMMATION DU PARALLELISME.

"Although "bugs" in programs are often embarassing to a programmer, if we are to learn from our mistakes we should record them and tell other people about them instead of forgetting them !"

D.E. KNUTH

1. Interblocage pour l'accès au télétype de l'opérateur.

a) Lorsqu'une erreur est détectée par programme dans une primitive du moniteur, fonctionnant interruptions masquées, la procédure suivante est appliquée :

- impression d'un message M sur le télétype de l'opérateur ,
- appel d'un moniteur "stand alone" auquel l'opérateur peut demander une impression ou une modification des données du moniteur.

L'impression du message se fait comme suit dans une procédure appelée BASTRINGUE :

```
procédure BASTRINGUE ;
        begin
        integer n, LIMIT ; boolean OK ;
        OK := false ;
        n := 1 ;
        comment : LIMIT sert à compter le temps d'impression d'une ligne ;
        comment : attendre la fin de la ligne en cours de frappe et lancer
                  le transfert du message M aussitôt après ;
        repeat
            begin
            TIO (OK) ; comment : si le coupleur est libre, OK est mis à
                                 true ;
            n := n+1 ; if n > LIMIT then faire siffler le 10070 ; return ;
            if OK then
                 begin
                 OK := false ;
```

```
                SIO(OK) ; comment : demande le transfert . OK est mis à
                          true s'il est accepté ;
                end ;
            end ;
        until OK ;
        comment : attendre que l'impression du message M  se termine
                  normalement ;
        OK := false ;
        for n := 1 step 1 until LIMIT do
                  begin
                  TIO(OK) ; comment : détecte la fin de transfert ;
                  if OK then return ;
                  end
        faire siffler le 10070 ; comment : le haut parleur du 10070 est
        utilisé comme moyen extrême pour attirer l'attention de l'opérateur ;
        end ;
```

La procédure BASTRINGUE ne sert que pour avertir des erreurs fatales du système. Pour éviter de la détériorer par un autre usage, la gestion du télétype de l'opérateur est faite d'une autre manière. C'est le cas lorsqu'il est utilisé pour assurer la conversation de l'opérateur avec le système d'entrée-sortie, avec le travail de fond, ou avec le moniteur. Le télétype est alors associé au processus SERVANT (voir 3e partie, chap. 3.4.3) et le parallélisme de l'unité centrale et de l'unité d'échange est utilisé. On a schématiquement :

```
SERVANT : begin
          P(SCONSSERVANT) ;
          SIO ;
          P(SITSERVANT) ;
          goto SERVANT ;
          end
```

Le sous programme d'interruption associé à l'interruption de fin de transfert effectue :

```
          TIO ; comment : cette instruction libère le coupleur du
                          télétype ;
          V(SITSERVANT) ;
```

b) Où est l'erreur ? Le masquage des interruptions, avant l'appel de BASTRINGUE, peut empêcher l'exécution du sous programme d'interruption et la libération du télétype occupé par le servant. Dans BASTRINGUE, le processus boucle sur le 1er TIO jusqu'au coup de sifflet final. C'est un exemple d'étreinte fatale :

Bastringue possède l'unité centrale et demande le télétype,
le servant possède le télétype et demande l'unité centrale pour le libérer.

C'est aussi un exemple du danger qu'il y a à mêler les emplois des interruptions et des sémaphores.

C) Corrections. Ce sont des corrections classiques de l'interblocage :

- réquisition d'une ressource : Bastringue peut prendre autoritairement le télétype en commençant par un HIO
- attribution globale des deux ressources. Pour cela, encadrer les actions sur le télétype par une section critique. On aurait :

  - dans le cas d'erreur du système :

```
démasquer les interruptions
P(SMUTEX) ;
masquer les interruptions
BASTRINGUE ;
appel du moniteur "stand alone" ;
V(SMUTEX) ;
```

  - dans le SERVANT :

```
P(SMUTEX) ;
SIO ;
P(SITSERVANT) ;
V(SMUTEX) ;
```

Nous avons adopté la 1ere solution, parce qu'elle permet de masquer immédiatement les interruptions.

2. Trappe ou incident dans une section critique.

Les réactions du système aux incidents ou aux trappes (déroutement) se produisant dans des sections critiques du moniteur ont été une bonne source d'erreurs. Ces erreurs ont résulté, dans ces cas, d'une analyse insuffisante du comportement du système en face de ses erreurs ou des erreurs des usagers pendant qu'ils utilisent des procédures du système. Pour bien comprendre ce problème délicat nous allons présenter deux solutions auxquelles nous avons abouti et quelques erreurs survenues.

Pour éviter toute confusion, nous appellerons incident tout emploi intempestif d'une procédure du système conduisant à un retour anormal, et erreur toute erreur de logique ou de programmation de notre part.

a) Réaction à un incident dans une procédure simple du système

Nous appelons procédure simple du système, une procédure du système appelée par une instruction CAL et se déroulant entièrement en adresses physiques. Rappelons que l'instruction CAL se comporte comme une trappe vers le moniteur et qu'elle peut être utilisée pour changer les modes de l'unité centrale (maître ou asservi, avec ou sans topographie) et lancer l'exécution de procédures du moniteur. Les paramètres et les résultats de ces procédures sont transmis dans les registres généraux.

Exemples de procédures simples du système : créer un processus, P, V, ouvrir un fichier, coupler, fermer un fichier.

Les paramètres de ces procédures sont des valeurs qui sont contrôlées avant leur utilisation. Tout paramètre erroné entraine un retour anormal pour incident. Ce retour varie selon le type de processus qui a effectué le CAL :

- Si c'est un processus du moniteur résident, l'incident correspond à une erreur du système. Un message est imprimé sur le télétype de l'opérateur et le système est arrêté.
- Si c'est un processus du moniteur virtuel (processus premier d'un usager, processus facteur de PTT), l'incident peut provenir d'une erreur du système ou d'une mauvaise commande d'un utilisateur. L'analyse doit en être faite par la procédure appelante. Le code condition, CC, de la machine est chargé à une valeur non nulle et le retour se fait normalement.
- Si c'est un processus quelconque d'un usager, l'incident provient d'une erreur de celui-ci. Il est inutile de laisser se poursuivre son programme. Un message d'erreur est préparé. Si l'utilisateur n'a rien précisé au début de son programme, c'est le cas standard et le processus premier de l'usager est réveillé. Il arrête tous les autres processus de l'usager, imprime le message d'erreur sur la console de l'utilisateur et revient à l'état de référence où il attend une nouvelle commande de l'utili-

sateur. Dans le cas non standard, l'utilisateur a précisé l'adresse de son programme où doit se faire le retour anormal au premier incident. Le processus premier n'entre pas en jeu.

Dans tous les cas, le CC indique le résultat du contrôle des paramètres. Il n'est nul que si tous les paramètres sont acceptés.

Le même schéma est utilisé en cas de trappe autre que le défaut de page, en assimilant la trappe à la détection d'un paramètre erroné. Une variante toutefois est nécessaire dans le cas d'un processus du moniteur virtuel. Le retour de trappe se fait via le processus premier, comme dans le cas standard d'un processus quelconque.

Dans ce schéma, le retour d'un CAL ou d'une trappe s'effectue par l'instruction LPS(RAB) où RAB indique un double registre contenant le mot état programme du processus à réactiver. RAB contient normalement l'adresse suivante du programme à exécuter. Il peut être changé en cas d'incident. Deux procédures sont utilisées :

```
comment : PROELU indique le nom du processus auquel est alloué l'unité
          centrale ;
procédure FINDEROUTEMENT ; comment : elle est appelée en cas de trappe, ou
                                     par RETOUR INCIDENT ;
    begin
    if PROELU ∈ moniteur résident then
        begin
        BASTRINGUE ; comment : voir exemple 1 plus haut. Les interrup-
                               tions sont masquées dans tout CAL ou
                               trappe.
        appel moniteur "stand alone" ;
        return ;
        end ;
    if (PROELU = processus quelconque) & (cas non standard) then
        RAB := adresse non standard
    else
        begin
        if PROELU ∈ moniteur virtuel then
            RAB := adresse dans processus premier
        else
            begin
```

```
            E.PS(S:ERREUR) ;
            E.VS(S:PREM) ;
            end ;
        end ;
end ;
comment : E.PS et E.VS se comportent comme les primitives P et V à cela
          près qu'elles sont appelées par une instruction BAL (et non
          CAL) et qu'elles n'activent ni ne bloquent de processus mais
          se contentent de mettre dans PROELU et RAB le nom et le mot
          d'état programme du processus à activer.

            S:ERREUR est un sémaphore de blocage en cas d'erreur ;
            S:PREM est le sémaphore privé du processus premier (voir
                  3ème partie, chapitre 2.7.2) ;
procédure : RETOUR.INCIDENT ; comment : elle est appelée dans le CAL
                                        seulement ;
    if PROELU ≠ moniteur virtuel then FINDEROUTEMENT ;
            Ce schéma ne pose pas de problèmes à ce niveau.
```

b) Réaction à un incident dans une procédure complexe du système.

Nous appelons procédure complexe du système une procédure du système appelée par une instruction CAL et se déroulant partiellement en mémoire physique, partiellement en mémoire virtuelle. Le passage par l'adressage virtuel est nécessaire pour avoir accès, via le défaut de page, aux données non résidentes du moniteur : données privées de chaque processus premier, catalogues des utilisateurs. La partie en virtuel de ces procédures est elle-même non résidente. Les paramètres de ces procédures contiennent parfois des adresses.

Une procédure complexe du système suit le schéma suivant :

- physique : contrôle des paramètres, action sur les données résidentes
- virtuel : action sur les données non résidentes
- physique : action sur des données résidentes, sortie du CAL.

La partie en virtuel doit se dérouler en mode asservi pour pouvoir utiliser le défaut de page. On passe de physique en virtuel par une instruction LPSD et de virtuel en physique par un CAL.

Les procédures complexes agissent sur des données propres à un usager. Elles sont donc nécessairement toutes en exclusion mutuelle sur ces données. Un sémaphore par usager, S:ARRET permet de règler cette exclusion.

```
comment : initialement PRENEUR=NIL, S:ARRET=1, CRAB est quelconque. Ces
          données sont associées à chaque usager. PRENEUR ¬=NIL est un
          témoin d'exécution en virtuel
procédure : COMPLEXE
          if PRENEUR ¬=PROELU then comment : il s'agit d'une première
                                             entrée dans le CAL ;
               begin
               if paramètre erroné then
                    begin
                    RETOUR.INCIDENT ;
                    LPSD(RAB) ;
                    end
               E.PS(S:ARRET) ; comment : entrée  en section critique ;
               comment : voir commentaire en fin de procédure ;
               if PRENEUR ¬=NIL then LPSD(RAB) ;
               action en physique ;
               PRENEUR := PROELU ;
               CRAB := RAB ;
               RAB := adresse virtuelle de la suite de la procédure ;
               LPSD(RAB) ;
               end
          else
               begin comment : il s'agit du retour de la partie virtuelle ;
               action en physique ;
               PRENEUR := NIL ;
               RAB := CRAB ;
               E.VS(S:ARRET) ;
               if incident signalée then RETOUR.INCIDENT ;
               LPSD(RAB) ;
               end ;
comment : l'action de E.PS est la suivante. Si aucun processus n'utilise
          le CAL (c'est-à-dire si PRENEUR=NIL), le sémaphore est mis à zéro
          pour arrêter tout nouvel appel. Si le CAL est déjà utilisé, le
          processus appelant est mis dans la file d'attente de S:ARRET, un
          nouveau processus est choisi (dans ESOPE, il en existe toujours
          au moins un, le processus FINAL), son indice est mis dans PROELU
```

et son PSD dans RAB. Son activation est à la charge du CAL dans lequel est exécuté E.PS, d'où la raison de l'instruction conditionnelle suivant E.PS(S:ARRET). Lorsque le CAL sera disponible, le processus bloqué sera libéré par E.VS et il exécutera à nouveau l'instruction CAL ;

Lorsque la procédure se déroule en adresses virtuelles, diverses causes peuvent arrêter son cours :

- la découverte par programme de paramètres incorrects
- le déroutement pour défaut de pages
- l'utilisation d'une adresse erronée passée comme paramètre et causant un déroutement pour page non couplée
- l'appel erroné d'une procédure simple du système. A cause de de l'exclusion mutuelle, toute utilisation récursive des procédures complexes est interdite, car elle conduirait à un blocage du système.

Le 1er cas est traité en revenant en physique avec un message d'incident. Le 2e cas est traité normalement et la procédure est réactivée lors de l'arrivée de la page manquante. Le 3e et le 4e cas causent un appel aux procédures FINDEROUTEMENT et RETOUR.INCIDENT pour le compte du déroutement ou de la procédure simple. Il en résulte une sortie de section critique sans que pour autant l'action E.VS(S:ARRET) soit faite. Aucune nouvelle procédure complexe ne peut plus s'exécuter pour cet usager. Il faut modifier en conséquence les procédures d'erreurs.

D'autre part les procédures d'erreurs transmettent l'adresse de l'instruction où l'erreur s'est manifestée. Or cette instruction fait partie de la partie virtuelle de la procédure complexe et est inconnue du programmeur. Il faut pouvoir fournir l'adresse d'appel de cette procédure complexe, de manière à permettre au programmeur de vérifier les paramètres d'appel.

Les procédures de sortie d'erreur deviennent dans ces conditions :

```
procédure FINDEROUTEMENT ; comment : appelée par trappe ou par
                                     RETOUR.INCIDENT ;
    begin
    if PROELU ∈ moniteur résident then
        begin
        BASTRINGUE ;
        appel moniteur "stand alone" ;
        return ;
        end ;
    if PROELU = PRENEUR then
        begin
        comment : PROELU exécute la partie virtuelle d'une procédure
                  complexe ;
        message d'erreur := if PROELU ∈ moniteur virtuel then RAB else
                                                              CRAB ;
        E.VS(S:ARRET) ; comment : sortie de section critique de la
                                  procédure complexe ;
        PRENEUR := NIL ;
        RAB := CRAB ;
        end ;
    if (PROELU = processus quelconque) & (cas non standard) then
        RAB := adresse non standard
    else
        if PROELU ∈ moniteur virtuel then
           RAB := adresse dans processus premier
        else
            begin
            E.PS(S:ERREUR) ;
            E.VS(S:IREM) ;
            end ;
    end ;
procédure RETOUR.INCIDENT ; comment : appelée par le CAL seulement ;
        if PROELU ∉ moniteur virtuel then FINDEROUTEMENT
        else
            begin
            E.VS(S:ARRET) ; comment : sortie de section critique ;
            PRENEUR := NIL ;
            end ;
```

La situation est encore plus compliquée à cause du parallélisme des processus d'un même usager. Lorsqu'un processus exécute une procédure complexe du système, après de multiples vérifications il reçoit le droit d'accéder au catalogue du système en le couplant dans sa mémoire virtuelle. Or celle ci est commune à tous les processus d'un usager, qui, si on ne prenait pas de précautions, pourraient alors écrire n'importe quoi dans le catalogue. Pour éviter cette conséquence, deux remèdes peuvent être appliqués :

- le plus brutal est de supprimer tout parallélisme chez l'usager pendant l'exécution d'une procédure complexe du système
- une autre méthode consiste à associer les droits d'accès à une page de mémoire non plus à un usager mais à un processus d'un usager. La seule façon de le faire, sans abandonner la notion d'usager, est d'utiliser les clés et verrous d'écriture en mémoire physique qui sont un dispositif câblé du 10070. Comme il n'y a pas assez de clé pour en attacher une à chaque processus, nous avons introduit deux zones de mémoire virtuelle, une protégée par verrous d'écriture et une non protégée. Un processus ne peut écrire dans la zone protégée que s'il a la clé d'écriture. Cette clé est attribuée en permanence aux processus du moniteur non résident et est prêtée aux autres processus pendant la partie virtuelle des procédures complexes du système. Lorsqu'un processus qui a la clé d'écriture fait un défaut de page, la page physique utilisée pour charger la page manquante reçoit le bon verrou. Aussi, en dehors des procédures complexes, les processus quelconques ne peuvent pas y écrire.

Cette seconde méthode est la plus élégante, mais elle demandait des modifications assez compliquées dans l'allocateur d'ESOPE. Aussi en attendant de la faire, avons-nous utilisé la première méthode avec quelques vicissitudes que nous examinerons plus loin.

A ce stade, nous sommes arrivés, étape après étape, à une construction compliquée. Les erreurs que nous avons eues provenaient :

- de l'oubli de sortir correctement de la section critique
- de l'imbrication des procédures simples du système dans une procédure complexe du système et de l'ignorance de ces imbrications dans les procédures de sortie d'erreur
- et surtout de la méthode de construction de la solution pas à pas et de l'absence de réflexion sur le problème global.

c) Arrêt du parallélisme chez un usager pendant exécution d'une procédure comlexe du système

Dans ce cas E.PS(S:ARRET) et E.VS(S:ARRET) qui servent, dans la solution précédente, à garantir chez l'usager l'exclusion mutuelle des procédures complexes sont remplacés partout par :

- ARRET.DU.PARALLELISME, procédure qui arrête tous les autres processus de l'usager et les met dans la file du sémaphore S:ARRET ;
- E.DS(S:ARRET) qui libère tous les processus de la file du sémaphore S:ARRET.

Cette solution amplifie les erreurs (est-ce un avantage à exploiter lors de la mise au point, ou un inconvénient ?) car en présence d'erreur, on se retrouve selon les cas dans l'une des situations extrêmes :

- ou bien tous les processus demeurent bloqués et plus rien ne se passe,
- ou bien tous les processus sont actifs et la catastrophe arrive vite.

Enfin par paresse, par astuce et par souci d'économie de sémaphores, nous avons, pendant un temps, confondu les deux sémaphores S:ARRET et S:ERREUR (ce dernier est utilisé en cas d'erreur chez l'usager) que nous avions crus, à tort, exclusifs l'un de l'autre. Ce genre d'économie fut une amère leçon.

ANNEXE 2.

Liste des modifications simples à apporter à la machine CII 10070 pour améliorer les performances d'ESOPE.

1. Supprimer l'empilement des mots d'états programmes PSD lors d'une succession d'interruption et ranger le PSD à une adresse dépendant du processus interrompu et non du niveau d'interruption activé.
2. Ne pas perdre les interruptions successives sur le même niveau (ou à la même unité de liaison, dans le cas de l'interruption d'entrée sortie) lorsque ce niveau est masqué. Compter les interruptions envoyées pendant que le niveau est masqué.
3. Conserver le fonctionnement des protections virtuelles en mode maître, ce qui permettrait de faire de la pagination à la demande, même en mode maître.
4. Introduire pour chaque page de la mémoire centrale un bit d'utilisation et un bit d'écriture, ce qui faciliterait la gestion de cette mémoire.
5. Au lieu de l'adresse de l'instruction qui en est l'origine, obtenir après déroutement pour violation de protection virtuelle, le numéro de page virtuelle pour laquelle la violation s'est produite. On gagnerait 100 microsecondes à chaque déroutement.(soit 5 % du temps d'unité centrale).
6. Introduire une clé et un verrou d'écriture en virtuel, comme il en existe pour l'accès à la mémoire physique. Ceci permettrait de protéger les processus d'un usager entre eux.
7. Permettre la lecture de la mémoire topographique et du système d'interruption ce qui faciliterait la mise au point du système.

Par ailleurs, il serait logique de ne pas utiliser le code condition CC comme paramètre d'appel d'une instruction comme pour STM et LM, ni de l'écraser stupidement dans un XPSD associé à un CAL et de réserver son emploi comme valeur résultat de certaines instructions. Ceci permettrait de généraliser cet emploi comme valeur résultat de procédures.

BIBLIOGRAPHIE COMMENTEE

1. Bibliographie sur les systèmes HALIOTIS et ESOPE.

DERVILLE, KAISER, PEIROTES, TELLIER. Le système HALIOTIS. RIRO 1ère année n° 6 (novembre 1967).

CAE. Manuel de référence de l'unité centrale CAE 133, note technique 1493.

IRIA. Le projet ESOPE. Bulletin de l'IRIA n° 1 (1969).

CII. Ordinateur 10070, manuel d'exploitation, note technique 2444.

BETOURNE, BOULENGER, FERRIE, KAISER, KOTT, KRAKOWIAK, MOSSIERE. Process management and resource sharing in the multiacces system ESOPE. Second ACM symposium on operating system principles (oct. 1969) CACM 13,12 (décembre 1970).

BETOURNE, BOULENGER, FERRIE, KAISER, KRAKOWIAK, MOSSIERE.
Présentation générale du système Esope.
Notion d'espace virtuel dans le système Esope.
Allocation de ressources dans le système Esope.
3 articles présentés au congrès d'informatique AFCET (setp. 1970).

BLAIZOT, BLAIZOT, LORHO, VATOUX, Organisation du compilateur interpréteur CPL1. Congrès AFCET (sept. 1970).

KAISER. Exemple d'une structure adaptée au partage de temps. DRME : Ecole de printemps sur les structures de calculateurs (avril 1971).

BETOURNE, FERRIE, KAISER, KRAKOWIAK, MOSSIERE. System design using parallel processes. Congres IFIP (août 1971).

FERRIE, MOSSIERE. Esope: gestion des processus et partage des ressources. cahier IRIA n° 8 (janvier 1972).

BAUDET. Manuel de Editer. Note IRIA/ESOPE/B/01.

BOULENGER. Manuel du LP 10070. Note IRIA/LP/B/01.

BOULLIER, GROS, JANCENE, LEMAIRE, PRUSKER, SALTEL. "METAVISU", a general purpose graphic system. IFIP conference on graphic languages, Vancouver, Canada (mai 1972).

BAUDET, FERRIE, KAISER, MOSSIERE. Entrées-sorties dans un système à mémoire virtuelle. Congrès AFCET (novembre 1972).

BETOURNE, KRAKOWIAK. Simulation de l'allocation de ressource dans un système conversationnel à mémoire virtuelle paginée. Congrès AFCET (novembre 1972).

KAISER. Quelques problèmes de parallélisme et leurs solutions par sémaphore. Société mathématique de Bretagne. Actes des séminaires de Rennes 1972.

KRAKOWIAK. Conception et réalisation de système à accès multiples : allocation des ressources. Thèse à paraître.

2. Bibliographie générale

** intéressant*

*** mérite un détour*

**** vaut le voyage*

Guide Michelin

* ALEXANDER. Time sharing supervisor programs. International seminar on advanced programming 1,4 Jerusalem 1969.
Comparaison de quatre systèmes : MTS, Multics, CP/67 et TSS/360.

* ANDERSON, BERTAN, CONN, MALMQUIST, MILLSTEIN, TOKUBO.
Design of a time sharing system allowing interactive graphics. Proc ACM nat. conference 1968.
Petit système sur XDS SIGMA 7. Introduction de l'idée du couplage.

** ANONYME. Principes des systèmes d'exploitation des calculateurs, livre à paraître.
Exposé des fonctions des systèmes d'exploitation et des concepts qu'il est utile d'introduire pour leur analyse et leur construction.
Ouvrage collectif, réalisé par un groupe de travail AFCET, auquel participent des membres de l'équipe ESOPE, dont l'auteur de cette thèse. Destiné aux étudiants de D.E.A.

** ARDEN, GALLER, O'BRIEN, WESTERVELT. Program and adressing structure in a time sharing environment, JACM 13,1 (janv. 1966).
Méthode d'appel de procédure et de passation de paramètres conservant la réentrance dans un système segmenté.

* ARSAC. Les systèmes de conduite des ordinateurs.Dunod 1968. cours de maîtrise de l'institut de programmation de l'Université de Paris VI.

** BELADY. A Study of replacement algorithms for a virtual storage computer.
IBM Syst. Journal 5,2 (1966).
Première étude statistique du problème de remplacement de page dans un système paginé. Introduction d'un modèle simple de comportement de programme.

* BENSOUSSAN, CLINGEN, DALEY. The Multics virtual memory. CACM 15,5 (mai 1972).
Motivation et réalisation de l'adressage par segments. Structure des catalogues dans MULTICS.

** BRYAN. JOSS : 20 000 hours at a console, a statistical summary.
AFIPS Vol.31 FJCC 1967.
D'intéressants résultats numériques sur le fonctionnement d'un tout premier système conversationnel.

*** BRAWN, GUSTAVSON.Program behaviour in a paging environment.
AFIPS V.33 (FJCC 1968).
Premières mesures de comportemement de programme dans un système paginé.
Comparaison de l'influence des algorithmes de remplacement et du style de programmation.

** CLEARY. Process Handling on Burroughs B6500. 4th Australian computer conference. Adelaïde 1969.
Présentation des mécanismes mis en place dans la Burroughs B6500.

** COFFMAN, DENNING. Computer operating systems : modeling and analysis.
livre à paraître.
Première synthèse des divers modèles mathématiques établie pour analyser les systèmes d'exploitation ou certaines de leurs fonctions.

* COFFMAN, ELPHICK, SHOSHANI. System deadlocks. Computing surveys, 3,1 (juin 1971).
Panarama des méthodes de prévention et de détection du blocage.

* COFFMAN, VARIAN. Further experimental data on the behaviour of programs in a paging environment.CACM 11,7 (juillet 1968).
Analyse expérimentale du comportement de programmes en faisant varier la taille des pages, la taille de mémoire allouée et l'algorithme de remplacement.

* CONWAY. A multiprocessor system design.AFIPS V.24 (FJCC 1963).
Une toute première description d'un multiprocesseur et des problèmes de parallélisme que cela soulève.
Intérêt historique.

*** CORBATO. Sensitive issues in the design of multi use systems, MO 105 MIT dec 1968.
Suite de conseils plaisants et sensés tirés de l'expérience que l'auteur a acquise dans le projet MULTICS

*** CORBATO, VYSSOTSKY. Introduction and overview of the Multics System. AFIPS V. 27 (FJCC 1965).
Un classique.

** COSINE COMMITTEE. An undergraduate course on operating systems principles 1971. Programme d'enseignement étudié très en détail par une commission formée de DENNING, DENNIS, LAMPSON, HABERMAN, MUNTZ, TSICHRITZIS.

* CRISMAN (Editor). The compatible time sharing system : a programmer's guide MIT Press 1963.
Description et specification de CTSS.

** DALEY, DENNIS . Virtual memory, processes and sharing in multics CACM 11,5(mai 1968).
Description des mécanismes cablés et programmés permettant de réaliser une mémoire virtuelle segmentée et de partager des segments.

** DALEY, NEUMANN . A general purpose file system for secondary storage . AFIPS V. 27 (FJCC 1965).
Un autre classique.

* DENNING. The working set model for program behaviour.CACM 11,5 (mai 1968).
Etude d'un modèle de comportement de programme indépendant du type de machine sur lequel il s'exécute.

** DENNING. Third generation computer systems. Computing surveys 3,4 (dec 1971).
Panorama des aspects communs aux systèmes d'exploitation contemporains et des bases d'une théorie de tels systèmes.

* DENNING. Thrashing : its causes and prevention.AFIPS V. 33 (FJCC 1968).
Définition et explication de l'écroulement des systèmes paginés. Présentation de quelques méthodes présentives.

* DENNING. A statistical model for console behaviour in multi user system computers.CACM 11,9 (sept. 1968).
Introduction du modèle de console virtuelle.

*** DENNIS, VAN HORN. Programming semantics for multiprogrammed computations.CACM 9,3 (mars 1966).
Mécanismes nécessaires pour la segmentation, la protection, le partage, le parallélisme et la multiprogram-

mation. Concept de "capability"
Interêt historique.

* DENNIS. Concurrency. Advanced course on software engineering, Munich 1972.
Présentation des réseaux de Pétri et leur utilisation pour décrire un système.

** DETLEFSEN, MORRIS. A virtual processor for real time operation, in software engineering V.1 (TOU edit.) 1969.
Présentation de la structure du système d'exploitation de la machine MU5 construite à Manchester. Une partie des mécanismes sont cablés.

** DIJKSTRA. Solution of a problem in concurrent programming control. CACM 8,9 (sept. 1965)
Première solution publiée du problème de l'exclusion mutuelle, programmée en ALGOL standard sans instructions spéciales du type LOCK, UNLOCK ou TEST AND SET.

*** DIJKSTRA. Cooperating sequential processes. In programming languages, Genuys, Academic Press 1968.
Présentation de processus parallèles et de problèmes de synchronisation. Introduction des primitives P et V.

** DIJKSTRA. The structure of the THE multiprogramming system. CACM 11,5 (mai 1968).
Système d'exploitation utilisant les notions de processus et de niveau er organisé de manière à établir à priori sa validité.

* DIJKSTRA. Hierarchical ordering of sequential processes. Acta informatica 1971.
Problème des philosophes aux spaghettis.

* GIRAULT. Un des systèmes de multiprogrammation réalisés à l'Institut de programmation de la Faculté des Sciences de Paris. RIRO B.2. (1971).
Système reposant sur l'idée d'un moniteur centralisé alimenté uniquement par une file d'attente de commandes déposées par ses interlocuteurs.

*** HABERMANN. Prevention of system deadlocks.CACM 12,7 (juillet 1969).
Défintion du problème du blocage. Méthode de protection reposant sur la création d'une suite d'états sains.
Algorithme du banquier.

** HABERMANN. On the harmonious cooperation of abstract machines. Thèse, Math. dep.
Technological Univ. Eindhoven Pays Bas (1968).
Formalisation de processus et application au problème de synchronisation.

* HANSEN. RC 4000 software multiprogramming system, A/S Regnecentralen, Copenhague (avril 1969).
Description du système RC 4000 : principes, organisation caractéristiques.

** HANSEN. The nucleus of a multiprogramming system.CACM 13,4 (avril 1970).
Specifications d'un noyau minimum pour un systeme multiprogrammé. Orienté vers la communication entre processus parallèles à l'aide de messages gérés par le système.

* HANSEN. A comparaison of two synchronizing concepts. Acta Informatica 1,3 (1972).
Introduction d'une notation permettant d'exprimer clairement la notion de section critique.

* HAVENDER. Avoiding deadlock in multitasking systems.
I B M Syst J 2,7 (1968).
Indique quelques problèmes de blocage apparus dans l'OS/360 et propose l'utilisation des ressources dans un ordre fixe comme prévention du blocage.

* HELLERMAN. Some principles of time sharing scheduler strategies.
IBM Syst. J. n° 2 (1969).
Classification d'un certain nombre de politiques d'allocation de ressources.

* HOARE. Towards a theory of parallel programming. International seminar on operating system techniques, Belfast 1971.
Presentation d'une ébauche de théorie du parallélisme et d'une notation pour la section critique.
Traitement de quelques exemples.

** HOLT. Comments on prevention of system deadlocks. CACM 14,1 (janvier 1971).
Deux remarques pertinentes sur l'utilisation partielle de l'algorithme du banquier.

** KILBURN, EDWARDS, LANIGAN, SUMNER. One level storage system.
IRE Trans EC 11 (avril 1962).
Première description détaillée du mécanisme de pagination de l'ATLAS.

* KNUTH. Additional comments on a problem in concurrent programming control. CACM 9,5 (mai 1966).
Commentaires sur l'article de Dijkstra des CACM 8,9 (sept. 1965).

*** KNUTH. The art of computer programming. Addison Wesley 1968.
Une encyclopédie. Le manuel de "taupe" de l'informatique.

* KUEHNER, RANDELL. Demand paging in perspective. AFIPS V. 33.
(FJCC 1968).
Présentation des fonctions à remplir pour la gestion de mémoire en page à la demande. Intérêt historique.

* LAMPSON. A scheduling philosophy for multiprocessing systems. CACM 11,5 (mai 1968).
Présentation des problèmes de synchronisation et d'une solution autre que les sémaphores.

*** ORGANICK. The Multics system : an examination of its structure. MIT Press 1972.
Présentation pédagogique des principes et de la construction du système MULTICS.
Exposé d'un système très complexe.

* PATEL. Basic I/O Handling on Burroughs B6500. Second ACM Symposium on Operating System Principles, Princeton 1969.
Présentation des mécanismes utilisés pour les entrées-sorties.

** SALTZER. Traffic control in a multiplexed computer system.MAC TR 30 (thesis) juillet 1966.
Mécanismes de communication entre processus. Principes de réalisation dans le système MULTICS.

** SCHERR. An analysis of time shared computer systems.MAC TR 18 (thèse) MIT 1967.
Etude du fonctionnement du système CTSS.

** SCHILS. The load leveller.IBM research RC 2233 (oct. 1968).
Présentation d'une méthode de controle du degré de multiprogrammation dans un système. Résultats expérimentaux sur le M44/44X.

* VANTILBORGH, VAN LAMSWEERDE. On an extension of Dijkstra's semaphore primitives. MBLE report R 192 (avril 1972).
Introduction et utilisations des opérations P(n,s) et V(n,s).

*** VYSSOTSKY, CORBATO, GRAHAM. Structure of the multics supervisor. AFIPS V.27 (FJCC 1965).
encore un classique.

*** WATSON . Time sharing design concepts,Mc Graw Hill(1970).
Premier ouvrage important consacré au partage de ressources. Bonne présentation des mécanismes d'adressage et de protection.

* WHITFIELD. The organization of the University of Edinburg time sharing system , International seminar on advanced programming system 2,5 Jerusalem 1968 .
Présentation du système EMAS. Première utilisation de la notion de catégorie pour l'allocation de ressources.

*** WILKES . Time sharing computer system. Mac Donald 1968 .
Petit ouvrage d'introduction aux problèmes posés par la réalisation des systèmes à partage de ressources.

** WILKES . Automatic load adjustment in time sharing system. Cahier du seminaire structure et programmation de l'IRIA 1971.
Présentation et analyse du modèle du régulateur de charge utilisé dans CTSS. Etude de sa stabilité.

* WIRTH . On multiprogramming, machine coding and computer organization , CACM 12,9 (septembre 1969).
Présentation d'une programmation en PL 360 des sémaphores et d'une fonction d'entrée sortie sur IBM 360.